TITLE:

STUDIES ON PROGRAMMING SYSTEM FOR BLOCK DIAGRAM SIMULATION AND ITS APPLICATIONS(Dissertation_全文)

AUTHOR(S):

Niimi, Yasuhisa

CITATION:

Niimi, Yasuhisa. STUDIES ON PROGRAMMING SYSTEM FOR BLOCK DIAGRAM SIMULATION AND ITS APPLICATIONS. 京都大学, 1969, 工学博士

ISSUE DATE:

1969-03-24

URL:

https://doi.org/10.14989/doctor.r1395

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

主論文　I

STUDIES ON PROGRAMMING SYSTEM FOR BLOCK DIAGRAM SIMULATION AND ITS APPLICATIONS

BY

YASUHISA NIIMI

Thesis, Kyoto University

December, 1967

STUDIES ON PROGRAMMING SYSTEM FOR BLOCK DIAGRAM SIMULATION AND ITS APPLICATIONS

BY

YASUHISA NIIMI

Thesis, Kyoto University

December, 1967

DOC
1968
6
電気系

PREFACE

It is often said that the first half of the 20th century is the age of energy and the latter half is the age of information. Information processing involves the effective handling of numerical data in a broad sense. However our principal interest consists in the information processing in a narrow sense which is closely related to the study of intellectual behaviors of human beings including recognizing process of two demensional patterns, speech sounds and sequences of semantic symbols or language, learning process and thought process.

In these fields, not only solving the mechanism of them but also the physical realization of them are pursued. A number of psychological and macroscopic experiments have been conducted to study them. However it has been difficult to perform physiological and microscopic experiments using human beings as subjects from the nature of the experiments, though they seem to be inevitable for solving the mechanism of intellectual behaviors of human beings. Therefore the structures assumed under hypotheses have been investigated regarding a psychological phenomenon as a response of a black box. These experiments have been carried out not only by means of physical devices but also with the aid of digital computers. The use of digital computers seems to increase more and more in the future because of their flexibility.

In engineering, a number of special machines for character recognition and speech recognition have been designed and studied. In these studies, digital computers are also used effectively for the examination of methods and the checking of designs. The co-operation of a special machine and a computer has been considered positively of late.

Thus digital computers play the important role in fields of information processing. It can be said that the study of information processing is greatly due to the development of digital computers. The both are related so closely that the development of computers seems to be the development of information processing itself.

Although a digital computer is often overestimated so that it is called an artificial brain, its ability depends deeply upon the program which is a schedule of its operations. What it can recognize is the sequence of meaningless symbols to man. It is almost impossible for man to handle these meaningless symbols on a large scale. Thus various kinds of programming languages have been proposed and developed. However, as fields for computer application become wider, it is more difficult to describe various kinds of problems by one language whatever general properties it has. Therefore, a great number of problem oriented languages or simulation languages have been developed.

In this thesis, a simulation language which can be used effectively and conveniently for simulation problems of pattern recognition systems and communication network systems is studied. For the purpose of testing advantages of the language developed, it is applied to speech analyses. The appendix is the reference manual containing detailed descriptions of the language.

December 1967

Yasuhisa Niimi

CONTENTS

CHAPTER 1

Introduction

In Pennsylvania in 1946 the electronic numerical integrator and computer ENIAC, worked hard to improve the rate of hits of anti-aircraft guns while struggling against the heat radiating from eighteen thausand vacuum tubes. Since that time there has been a surprising development in digital computers which has been associated with the rapid advance in the techniques of the elemental parts comprised in them. Now the new generation of the computer, that is, the generation of man-machine co-operation is upon us. Although various studies have been made of its possible uses, the abilities of digital computers are especially important in those fields in which arithmetical operations play an important role. Several arithmetical formular languages have been developed to perform the arithmetical operations easily and effectively. They seem to have been incorporated in FORTRAN[1] and ALGOL[2]. Furthermore, FORTRAN and ALGOL are also to be incorporated in the more powerful general purpose language PL/1[3].

The increase of the speed and the memory capacity of the computer enables ideas previously rejected as impractical to be utilized. When Buffon evaluated the value of π by falling a needle, as a postime of leisure hours, his method was perhaps of mild interest. However, who could have predicted that it would prove to be a powerful aid to numerical experiments? Indeed, the rapid progress of the computer enables a method similar to it, referred to as the Monte Carlo method[4], to be used effectively in numerical experiments in wide fields in combination

with the new concept of simulation. This combination of the statistical or probabilistic method and simulation, due to Von Neumann and Ulam, has opened the wide areas for computer application.

The definition of simulation in a broad sense means a desk experiment which is carried out using a physical, linguistic and mathematical model extracted from the actual object of study. In general there are the following three types of simulation[5]: all-computer simulation in which only computers are used; all-man simulation in which only the subjects are used; and man-machine simulation[6] which can be constructed by interactions between the computer programs and a group of the subjects. We will limit ourselves here to the first type, that is, all-computer simulation. This simulation in a narrow sense can be classified into the following two types.

The first are those cases in which no other means of experiments than computer simulation can be obtained. Since the object of study is closely related to such real problems as forcasting economic trends in a state, constructing a traffic control system in a city, etc., the practical testing cannot be carried out without accompanying dangers or possible economic losses.

The second are those cases in which it is desired to build up some system physically, although its specifications are not yet complete. The simulations supplement the imperfect parts of such systems under tentative assumptions until these assumptions can be finally verified. The imperfection may be concerned with parameters or structures of the system. Even in cases where the specifications are completed, the behavior of the system is examined by the computer simulation in order to discover whether or not the desired results will be achieved so that the

design may be infallible.

In general simulation can be shown schematically in Fig. 1-1. The system is the object of study, such as a traffic control system or the building up of a physical design as discussed above. Although the complexity of the system and the uncertainties of so many elements of it are such that it must be investigated with simulation, an investigator can extract some information from his knowledge of the system in order to check the results obtained from simulation. Algorithm or model can be formulated on some assumptions and simplifications from the logical structures underlying the complicated mechanism of the system. The algorithm, which should be referred to rather as a specification of the second type of simulation, is usually represented by flow charts, block diagrams, a set of equations, a set of sentences, etc.. Programming is to make a program to operate the computer according to this algorithm. Comparison of the results which are obtained by carrying out the program with what an investigator expects, feeds back the command to alter some of the parameters of the model or to modify the formulation of the object of study. Thus simulation is performed again by the use of the new model. The pattern of formulation, programming, execution, comparison and reformulation seems to be essential in simulation.

Why is it necessary and advantageous to perform computer simulation? In the first type of simulation there is no other means of investigating the behavior of the system. In the second type of simulation, the system could be studied by constructing experimental devices. However, in order that the model may be as close to such a system as the investigator expects, unknown portions of the model, that is, hiatuses in the design must be eliminated by repeating the experiments. It is

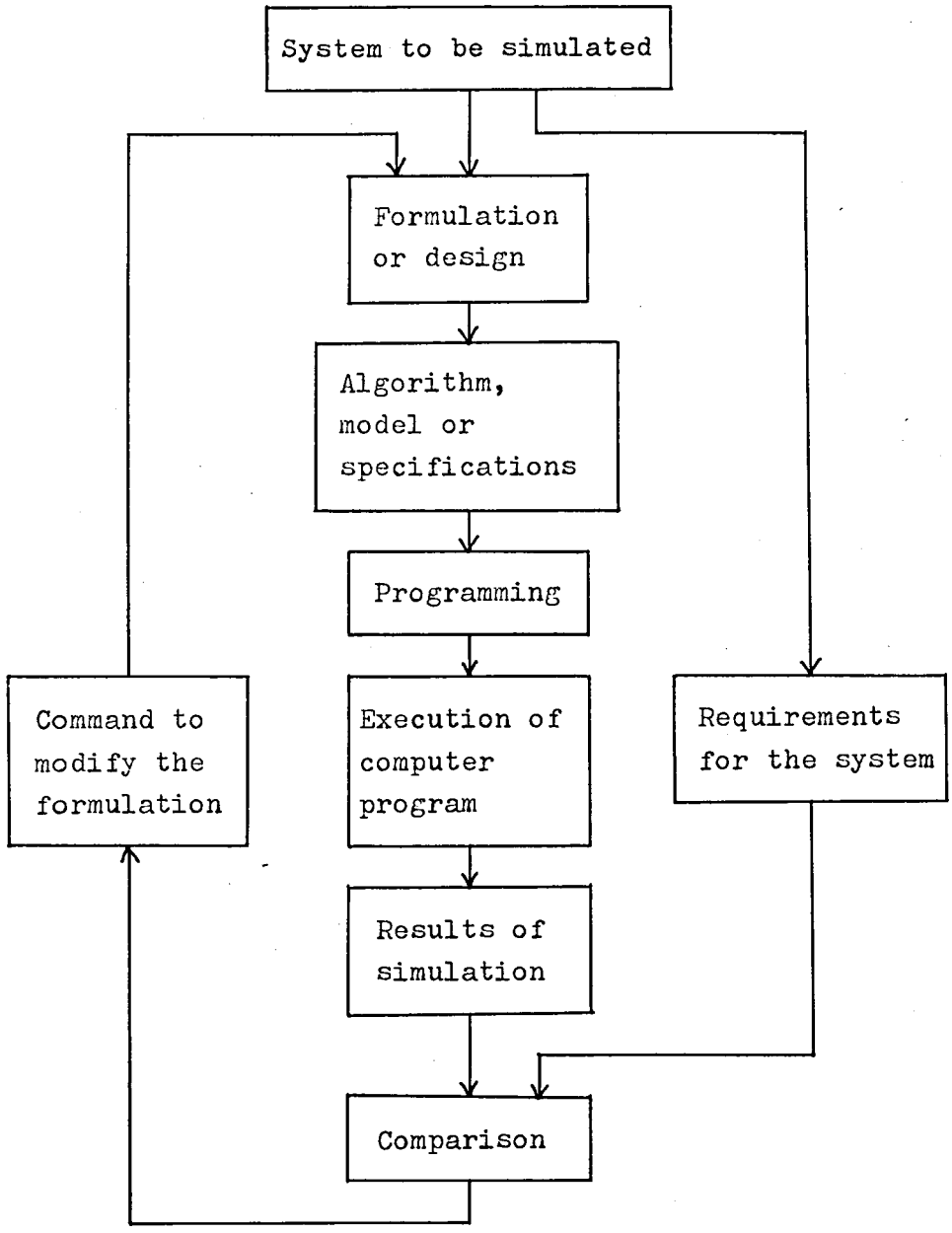


Fig. 1-1 Pattern of simulation.

very difficult to repeat a number of experiments for modifying a part of a device on account of the enormous time and expense involved. By contrast the elaborate programming system of the computer is so flexible that these can be done very easily and quickly. Indeed, one of the most profitable advantages of computer simulation lies in this fact. Furthermore a pure experiment that is not disturbed by noise is also possible if the time and expense required can be spared. The macroscopic and the microscopic behavior of the model can be observed by controlling the time scale in the computer. In formulating the model, which can greatly depend on intuitive factors, we can have more flexible ideas if we assume the use of computer simulation rather than proceeding on the basis of physical devices.

In general, the formulated models which may be described with a set of sentences, a set of the equations, flow charts, block diagrams, etc., are not always suited for programming with general purpose languages. If there be such a degree of similarity in the basic structure of the simulation models that they can be represented with one of the descriptive methods previously stated, a rapid method for systematic conversion into computer programs can be constructed. This is usually called a simulation language. Simulation languages have been proposed and developed for various fields.[7)-(15)] They not only afford ways to make simulation programs easily, but also have great influence on the formulation of simulation models. In order to achieve simulation effectively, therefore, it is important to know the features of various simulation languages before formulating a simulation model in this sense.

The study of a simulation language, KAIRO (this is a Japanese

word meaning a circuit), suited for the second type of simulation is described in the first half of this thesis. KAIRO lends itself best to simulations of information processing systems that are represented with block diagrams involving analog and digital signals.

Chapter 2 is a discussion of the differences between a general purpose language and a simulation language in simulation procedures. The difficulties that a general purpose language will encounter in simulation procedures are stated in detail. The need and the requirements for a simulation language are considered to overcome these difficulties.

Chapter 3 is devoted mainly to the description of the grammatical rules of KAIRO language. With KAIRO we can make easily a program to simulated a model represented with a block diagram. A statement of KAIRO which corresponds to one of the blocks included in it is expressed in terms of a name assigned to the block, a function code, parameters and the connection with other blocks. KAIRO also makes provision for repeating simulation runs while changing the parameters of the model, taking into consideration the essential procedure of simulation.

Chapter 4 is the description of the language processor which translates KAIRO into FORTRAN. One of the features of KARIO is that the order of statements specifying the model can be arbitrary and independent of the signal flow. This fact imposes a burden on the language processor. This difficulty can be solved by using the list structure and push-down memory.

In the latter half of this thesis, chapters 5 and 6, KAIRO is applied to studies of speech information processing. Chapter 5 is devoted to the study of infinitely clipped speech wave. The time length

of zero-crossing intervals is a continuous quantity. In the present study, it was quantized so that it was classified in one of the predefined number of intervals. The speech wave so transformed may be called a time quantized zero-crossing wave. The intelligibility of this wave was investigated. Furthermore, using data obtained in measuring the intelligibility, the confusion matrices were constructed, and it was found that vowels and fricative consonants were fairly stable, but that stop consonants and nasal consonants were much confused.

Chapter 6 is the study of the procedure for automatic extraction of FORMANT, one of the parameters which play an important role in the automatic recognition of vowels. This procedure was studied independently of the development of the simulation language, KAIRO, but was nevertheless done using the computer program, and some successful results were obtained. Realization of the procedure with hardware devices through analog techniques was tried in an attempt to form with the block types in KAIRO an estimate of the scale of the whole system. The supplementary experiments which were carried out using the block diagram constructed increased the reliability of the results previously obtained.

Appendix is the users' reference manual of KAIRO language (written in Japanese). It contains the detailed descriptions of the grammar of KAIRO language, especially the basic functions prepared in KAIRO language, the operation method, the meaning of error messages and some limitations mainly due to the scale of core memory of the available computer.

CHAPTER 2

Simulation Language

2-1 General purpose language

A simulation model can be described in various ways as stated in chapter one. By whatever form it is represented two kinds of entities, active and passive can be recognized in the simulation model. In the simulation of information processing systems they are exemplified by devices and signals. The interaction between these entities is that the signals are acted on by the devices and so changed. The simulation model must describe these changes and the time sequence of such interactions in the whole process.

A special program for simulating the model can be written in any one of the well-known general purpose languages such as FORTRAN, ALGOL or PL/1, or in one of the so-called simulation languages. Whichever is used, a general purpose language or a simulation language, the first attention has to be focused on how to represent the entities and how to express the interactions and their time sequence. It may fairly be said that the ease of programming and the range of the problems which can be treated depend on it. The alternative of a general purpose language offers the investigator maximum flexibility in the design and formulation of the mathematical model of the system being studied. Using the arithmetical, the logical, the control and the input-output statements of a general purpose language, he can write the program to specify the interactions one by one and the time advancing mechanism by means of his own programming skill. Furthermore, a general purpose language may be used

similarly to a simulation language. Instead of attempting to write a special program for each simulation model, it is built up from a set of subroutines already developed. Although such a method may require a great deal of programming time to develop all the subroutines, it enables the investigator to simulate a larger scale of simulation model conveniently. However it is inevitable that the range of simulation models which can be built up from a finite set of subroutines is limited. This method may be called a modular method.

So far as the simulation model, no matter how complicated it may be, is limited to that whose behavior can be described in terms of arithmetical and logical statements between algebraic variables, any algorithmic language is capable of expressing the required model in principle. However flexibility in these points is always associated with an increase in difficulty in programming. The problems to be investigated by means of simulation are generally those which have no analytical solutions or which cannot be expressed in an analytical form on account of complexity even if so. The difficulty which may be encountered in writing a program for complicated models, especially the dynamically controlled systems, lies in the control of the sequence which consists of an organization for advancing the time and ordering the interactions. It is left in the care of the programmer how to advance the time through the simulation and how to piece together the various blocks of program, which express the interactions occurring in the model, into a consistent order. This difficulty is the disadvantage of programming with a general purpose language.

The next thing that should be done after programming is debugging. During the first trial runs of the model, the programmer must obtain evidence that his model operates correctly in the computer. So far as

error checking procedure is concerned, there is the difference between a general purpose language itself and a modular method. With the modular method errors may only possibly occur in the main program which calls the elaborate subroutines to build up the simulation model. On the other hand, the possibility of committing an error will increase much more with a general purpose language because the programs corresponding to the subroutines in the modular method must be made from scratch. Any language processor of a general purpose language can check the programs for rule violations or capacity violations but as a rule, is not able to check for logical errors in simulation programs. In order to obtain any information to trace the fault, special statements must be inserted where they seem to be necessary.

As to output reports, a general purpose language still offers flexibility in their design whether they be of a graphical or a tabular nature. In this case flexibility is not associated with such difficulties as previously mentioned and so is very advantageous. Nevertheless it will serve to lighten a programming burden to provide subroutines to generate the graphical and the tabular output reports as well as subroutines to calculate the statistics such as the mean and the standard deviation and to form a histogram as done in the modular method.

Turning now to one of the basic elements in a simulation procedure we consider the modification of the simulation model. As construction of the model in the computer is closely related to the control of sequence, so is the modification of the model. Therefore it may be more possible to introduce new errors in the program by partially changing the structure of the model than by simply changing the parameters of the model.

In spite of its flexibility, a general purpose language, including the modular method, has been proved to involve great difficulty in the most important part of a simulation procedure, that is, the conversion of the model into the actual computer program.

2-2 Simulation language[16] [17]

It has been proved that it takes great efforts and much time to develop a simulation program by the use of a general purpose language alone, and therefore that the rapidity and flexibility associated with computer simulation can not be fully attained. This is because the problems to be studied by means of computer simulation have become larger in scale and more complicated in structure with the advance of the computer. We have considered the origin of such difficulties, and a number of simulation languages have been proposed and developed in order to overcome them. The range of problems which can be studied efficiently by the use of one of these simulation languages is, of course, narrower than those which can be dealt with by a general purpose language. For instance, GPSS[7] is best suited to certain types of scheduling and waiting line problems, DYNAMO[8] and SIMULATE[15] lend themselves best to simulations of large scale economic systems that are described by econometric models involving complex feed back mechanisms, and BLODIB[10] [18] accepts the sampled data systems described by a block diagram. These simulation languages aim chiefly at the practical utility of an easy conversion of a simulation model into a computer program at the cost of generality and flexibility.

In fact they have been developed with the following objectives in

mind[9],

1. To produce a generalized structure for designing simulation models.
2. To provide a rapid way of converting a simulation model into a computer program.
3. To provide a rapid way of making changes in the simulation model that can be readily reflected in the machine program.
4. To provide a flexible way of obtaining useful outputs for analysis.

Since the range of programs which can be studied by means of a simulation language is limited to some extent, some degree of similarity will arise in the basic structure of simulation models. Thus, because it is possible to extract the logical structure underlying the limited range of problems and certain concepts and operations which are used commonly in these, it is possible to give more structure to a simulation language than is possible to a general purpose language which has a simple nested block structure. In this sense simulation languages are a higher order of languages than the general purpose languages, though simulation languages are taken as particularizations of the general purpose languages in the sense that, however complicated the simulation model may be, it can be expressed in principle by means of a general purpose language.

The description of a simulation model consists of the specification of events which occur in the model and the specification of the interactions of these events and the time sequence in which these interactions occur. The events may be described conveniently in terms of such macro instructions as are provided in a simulation language as the basic operation common in the models rather than in terms of arithmetical, logical and control statements of a general purpose language. These macro

instructions correspond to the subroutines of the modular method outlined in the previous section. However it is almost impossible to design a language so that every facility required in the future is provided for. Consequently, the language should be made open-ended by allowing the user to design his own facilities in another language, for example, a general purpose language, or an assembly language, and to give these abbreviated names. This is known as the subroutine facility.

The complexity of the interactions of the events and the time sequence in which they occur depends primarily on the properties of the simulation model. For instance, in methods for moving the model through time, there are two types of time flow mechanisms in use today —— the fixed time increment programming method and the variable time increment method. Of course the latter method is more complicated than the former. A simulation language should be responsible for making a program related to these interactions and the time sequence. In particular the time sequence control should be made automatic because this is so difficult that it is a major shortcoming in a general purpose language.

Even if a simulation model is easy to convert into a computer program by the use of a simulation language, the possibility of the occurrence of error still remains. This is because the more complicated is the structure of the model, the more difficult it is to obtain any information manually to make certain that the model works correctly in the computer. It is desirable, therefore, to discover the bugs in the program automatically at early stage, and for it to be possible to obtain conveniently informations which is of use in discovering them.

With a general purpose language, rule violations and capacity

violations are only checked before the final simulation results are examined. On the other hand, the language processor of a simulation language should include a diagnostic or error checking program that can detect not only rule violations and capacity violations but also the primitive logical errors such as the existence of feed back mechanisms impossible to order in the sequence. This is one of the great differences between the two languages.

Output reports are very important because they are the material used to decide the results of the simulation and to issue the command to modify the simulation model. It is generally very troublesome to specify the style of output reports. In a simulation language, it is desirable to obtain several kinds of output reports such as a graphical output and a tabular printing conveniently without the detailed specification of the output formats. It seems self-contradictory to desire a wide range of output styles without the detailed specification of those styles. The solution is to define styles which seem to be required occasionally and to call for them by the abbreviated names.

CHAPTER 3

KAIRO language[19] [20] [21]

3-1 Introduction

The advantages and requirements of a simulation language have been considered in chapter two and have been compared with a general purpose language. Under these considerations a simulation language KAIRO (a Japanese word meaning a circuit) has been developed for the digital computer HITAC-5020, which is best suited to the problems concerning the simulation of an information processing system, including both analog and digital signals, such as communication networks, pattern recognition systems and speech analysis and synthesis systems. It is assumed that the digital circuits operate synchronously. A simulation model is described in the form of a block diagram drawn with a fixed set of predefined block types, the number of which is about fifty. In this sense KAIRO language may be called block diagram language. Each block type represents a specific action that is characteristic of some basic operation that can occur in an information processing system including both analog and digital signals. Connection between the blocks of the block diagram indicates the flow of signals through the model. The block diagram describing the simulation model is converted to a set of statements. One statement corresponds to one block in the block diagram and is expressed in terms of a label assigned to the block, a function code expressing block type, its input connections and parameters. The sequence of the statements may be arbitrary independently of the flow of signals through the model. It is also possible to repeat simulations

as changing some parameters of the model.

The KAIRO language processor accepts a source program consisting of a set of statements, reveals the sequence of the interactions between signals and devices described implicitly by the connection between the blocks and generates the program written in FORTRAN to simulate the model which it is desired to investigate. The generated FORTRAN object program is once punched on paper tape or cards, or written on a magnetic tape. It must further be processed by a FORTRAN compiler to execute a simulation run. (See Fig. 3-1)

The advantage of using FORTRAN as an interlanguage between KAIRO language and machine language enables FORTRAN statements to be inserted directly into a KAIRO source program. The flexibilities of FORTRAN are thereby combined with the facilities of KAIRO. This combination facility contributes to designing complex functions not provided as block type in KAIRO and various kinds of input and output routines. It is not desirable to confine to one machine a programming system troublesome to develop in the present situation in which the kind and the scale of available machines are rapidly developing. At least the FORTRAN object program of KARIO may operate in most other machines, which will probably have FORTRAN language processors. This is another reason for adopting FORTRAN as an interlanguage between KAIRO language and machine language. The disadvantage of doing so is the inefficiency of the final object machine program resulting from its passing through two language processors.

KAIRO is similar to GPSS[7], BLODIB[10][18] and other digital analog simulators[22][23][24] in that it is a block diagram language. It is mainly concerned with the problems of the signal processings which have less probabilistic characteristics, while GPSS is best suited

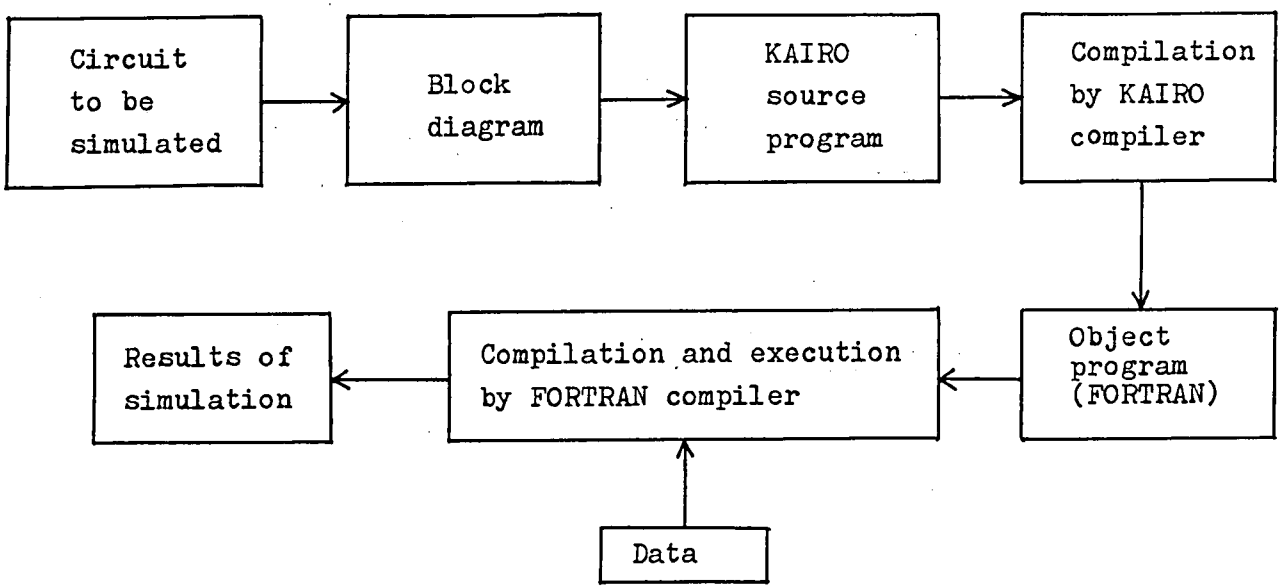


Fig.3-1 Block diagram of KAIRO system.

to certain types of scheduling and waiting line problems which are essentially probabilistic. Most digital analog simulators devote themselves to the digital simulation of analog computers, but not to pulse or logical circuits. KAIRO is very similar to BLODIB in the language structures, which played the part of a guiding model in developing KAIRO language. However, BLODIB is concerned with the sampled data system. In the simulation of continuous circuits whose input and outputs are band limited time functions, one must first design a pulse circuit whose output pulse would correspond to the sampled values of the desired output. On the other hand, KAIRO is rather similar to the digital analog simulators in that it attempts to simulate a continuous circuit directly, though the approximate computation is essential in the digital simulation. This difference will be apparent in the simulation of a linear network. It is performed by means of a z-transform in BLODIB, whereas KAIRO can simulate directly the linear networks represented in terms of a Laplace transform, $a/(s+b)$ or $(as+b)/\{(s+c)^2+d^2\}$. Although most of digital analog simulators generally use a refined integral procedure such as the second order Runge-Kutta method or predictor-corrector method, KAIRO adopts a simple trapezoid rule to reduce the computing time because most of our problems require a number of repeated cycles in a simulation. The error introduced by this simplification of the integral procedure can be tolerable, as will be exemplified in the section 3-5. KAIRO can handle the logical operations, but is rather inefficient in a large scale logical circuit compared with other special purpose logical simulators[25] because it has no provisions for partial word procedures.

3-2 Entities in KAIRO language

3-2-1 Signals and their representations

There are two kinds of entities, passive and active in KAIRO language. The passive entity is a signal flowing through a simulation model. The active one is an elemental device constructing the simulation model, which acts on the signal. There are the following four types of passive entities, that is, signals; (1) analog type, (2) integer type, (3) logical type and (4) trigger type. The analog type of signal is represented by a floating point number in the computer and other types of signals with fixed point numbers. The analog type of signal is a continuous time function, and the integer type of signal is the output of counters, which is used to control a circuit with multi-contacts. Although it is in practice represented by a binary signal of multi-channels, it may be idealized to the integer type of signal. Both logical and trigger type of signal are two valued logical functions. As their difference is shown in Fig.3-2, the logical type of signal represents the continuous time function, while the trigger type of signal represents the time discrete signal. It causes each block type to require the predefined types of signals and to produce the particular type of signal that the signals flowing in a simulation model have their particular types. The output of the block type that cannot be defined exactly as a delay line inherits that of input signal. The type mismatching, pertaining to input connection, that may possibly occur in the programming stage, will be discovered by the language processor.

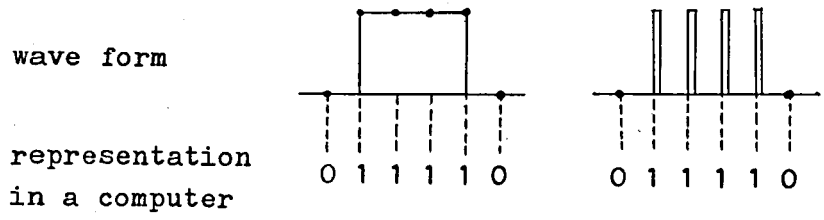


Fig. 3-2 Logical type(left) and trigger type(right) of signal.

3-2-2 Block type

The dominant type of entities in KAIRO language is an active entity, that is, a device. Connection of devices describes a block diagram, or a simulation model. On the other hand a signal does not appear explicitly in the description of the simulation model. A set of the elemental devices are provided as block types. Each block type represents a basic operation which is necessary to handle the kinds of signals described in the previous section. A list of all the available block types is shown in Table 3-1. Most block types have some input terminals and a signal output terminal. The number of inputs is fixed in some block types and in the other block types it may be arbitrary in principle, but the memory capacity of the available computer limits it to twenty-five in the present form of the language processor. Some block types have more than two output terminals. The number of outputs is fixed in some of them and may be arbitrary but is now limited to twenty five in the others. In the current form of the language, the delaying-type block, which plays an important part in constructing a

Table 3-1 List of Block Types.

Type	Function	Parameters	Inputs
INP1	Input box 1	Record length File name Clock time Device number	None
INP2	Input box 2	Record length Number of records Clock time	None
OUT1	Output box 1	File name Indicator of positioning Device number	Signal
OUT2	Output box 2	File name Indicator of positioning Device number Indicator of operation skipping	Signal
DEL	Delay line	Number of units delay	Signal
AMP	Amplifier	Gain	Signal
ADD	Adder	None	m-Signals
SUB	Subtractor	None	+ Input - Input
MLP	Multiplier	None	2-Signals
DIV	Divider	None	Dividend Divisor
BAT	Battery or bias	Bias	(Signal)
ACC	Accumulator	Reset level	Signal (Reset control)
MAX	Maximum circuit	None	m-Signals
MIN	Minimum circuit	None	m-Signals
HLD	Sample and hold	None	Signal Sampling control
MSW	Multi-switch	Off-level	Control m-Signals

Table 3-1 List of Block Types(continued).

Type	Function	Parameters	Inputs
DIS	Distributor	Off-level	Signal Control
DBC	Decibel converter	Gain	Signal
SQT	Square rooter	None	Signal
PCL	Peak clipper	Upper clipping level and/or Lower clipping level	Signal
BCL	Base clipper	Upper clipping level and/or Lower clipping level	Signal
QNT	Quantizer	Levels	Signal
LQT	Linear quantizer	Step size	Signal
ABS	Absoluter	None	Signal
AND	AND circuit	(Threshold)	m-Signals
EOR	Exclusive OR circuit	(Threshold)	2-Signals
IOR	Inclusive OR circuit	(Threshold)	m-Signals
NOT	NOT circuit	(Threshold)	Signal
CNT	Counter	Reset level Upper limit Lower limit	+Set signal (-Set signal) (Reset control)
SHR	Shift register	None	Signal (Shift control)
MMV	Monostable multivibrator	Pulse width	Set signal (Reset control)
FFL	Flip-flop	None	3-Signals
AMV	Astable multivibrator	Initial phase 2-Pulse widths	(Control)
SCH	Schmidt circuit	2-Thresholds	Signal
CIN	Coincidence circuit	2-Thresholds	Signal
TRG	Trigger circuit	None	Signal

Table 3-1 List of Block Types(continued).

Type	Function	Parameters	Inputs
FLT1	Filter 1 ($\frac{a_2}{s+a_1}$)	a_1, a_2 Sampling period	Signal
FLT2	Filter 2 ($\frac{a_3 s+a_4}{(s+a_1)^2+a_2^2}$)	a_1, a_2, a_3, a_4 Sampling period	Signal
RCF	Rectifier	Time constant Sampling period	Signal
COS	Cosine generator	Amplitude Frequency Phase Sampling period	None
RNG	Random noise generator	Initial phase Standard deviation	None
GEN	Function generator	(Period) Sample points Sample values	(Control)
PRT1	Printer 1	Number of I*-type inputs Number of A*-type inputs Starting point Pitch	m-Signals
PRT2	Printer 2	Number of I-type inputs Number of A-type inputs	Control m-Signals
SWT	Switch	None	Control 2-Signals
GRAPH	Graphical output	Maximum Minimum Starting point Pitch Number of sampling points	1-10 Signals
FBOX	FORTRAN box	Number of FORTRAN statements	m-Signals
DUM	Dummy box	None	Signal
CLOCK	Clock pulse generator	None	None

* A-type means an analog type of signal and I-type means other types of signals.

block diagram as will be stated in the next section, is a simple delay line only. Signal generators as well as output report generators are important in simulation. For this purpose the six block types are contained in the list. They are two block types to read data from a magnetic tape and from a paper tape, a function generator, a cosine wave generator, a random noise generator and an astable multivibrator. The output report generators are two printing routines, a graphical output routine and two magnetic tape handling routines.

3-2-3 Construction of a block diagram

KAIRO language has been developed to simulate a block diagram in which the analog and the digital signals can flow. The digital parts of the block diagram have been assumed to operate syschronously. In general the output of a block can depend on the present and all the past input signals. The block whose output depends only on the present input signal or signals is called memoryless. On the other hand, the block whose output depends on the present and the past input signals is called a block with memory. The time discrete, in other words, synchronous operation does not introduce errors into the output of a memoryless continuous network, while it will introduce some errors into the output of a continuous network with memory. For such a network to produce the correct output, the knowledge of only the input signals at the discrete time points is not sufficient. In other words, the output of the block with memory cannot but be determined approximately by the input signals at the current and all the past smapling points. If this approximate output can be accepted (cannot but be accepted, in fact), all the block, continuous as well as digital, to be simulated with KAIRO will thus

operate synchronously in this sense.

The block whose output is independent of the current input signal or signals is called a delaying-type block, and otherwise a nondelaying-type block. Only a simple delay line is specified as a delaying-type block in the list of block types, that is, Table 3-1. The output of a nondelaying-type block cannot be computed until all the inputs to this block have been determined, because the output of this type of block is a function of its current inputs. On the other hand the output of a delaying-type block can be determined independently of its present inputs. In other words, it is a function of (equal to, in fact) its past input. Therefore it must be computed before its input is over-written with a current value. If another delaying-type block refers to its output, the output of the former must be computed before that of the latter according to the rule just stated.

Thus the language processor cannot accept the block diagrams containing the following two kinds of closed loops:

(1) a closed loop with no delaying-type blocks;

(2) a closed loop with all delaying-type blocks.

Otherwise, the block diagram is deemed acceptable. If the first kind of closed loop has any physical meaning, its function can be replaced with any other block type or the acceptable combination of block types in most cases, as a simple amplifier can be substituted for an amplifier with a negative feed back loop that is not acceptable. Rejecting the second kind of closed loop would not make it difficult to construct a block diagram. The reason is that such a closed loop cannot produce any useful output because all the delaying-type blocks are initialized at zero. Whenever an acceptable block diagram can be drawn from the

block types in the fixed list, it can be simulated to the extent that the memory of the computer can accommodate.

3-3 KAIRO statement

KAIRO language has the two kinds of statements. One describes the blocks comprised in a block diagram to be simulated and the other describes the change of parameters of a block. These are called KAIRO statements.

3-3-1 Statement for the specification of a block

This statement consists of a label field, a function field, a parameter field and an input field. One statement describes one block in the diagram. The label field contains the label assigned to the block by the programmer. It is a string consisting of one to five alphabetic or numerical characters, the first of which must be alphabetic (in KAIRO language an alphabetic character means one of the capital letters from A through W, and the letters X, Y and Z are reserved for the variable names generated in the object program by the language processor and the programmer is forbidden to use them in order to prevent confusion between generated variables and programmer's labels). The label is regenerated, as it is, for a FORTRAN variable representing the output value of the block in the object program. One of the block types listed in Table 3-1 is contained in the function field as a function code. The parameters given to the block (such as a gain of an amplifier) are separated by commas and listed consecutively in the parameter field. They consist of the alphabetic, the integral and the

real parameter. The alphabetic parameter is a string composed of one to ten alphabetic or numerical characters, the first of which must be alphabetic, and is used for the name of an input or output data file. The integral parameter takes the same format as that of the I-type number and the real parameter takes the same format as that of the F-type number or the restricted E-type number in FORTRAN. The labels of the input blocks which must be the labels of other blocks are listed in the determined order in the input field, separated by commas. In some block types the number of the parameters and that of the input blocks must be fixed, but they may be arbitrary in other block types.

The KAIRO statement is punched on a card in the same format as that of the assembly language, HISAP[26], of HITAC-5020. In column 1 a blank or a character "C" is punched. The character indicates that the card should be interpreted as a comment card. The label field corresponds to the location field in the assembly language (columns 2 through 11) and the function field corresponds to the operation field (columns 12 through 18). The parameter and the input field are separated by closed quotation-marks and are listed consecutively in columns 19 through 72 corresponding to the address field. Each statement is generally punched on one card, but a provision for interpreting two or more cards as one statement is made. In this case the last comma on a card must be replaced by the continuation symbol " / " and the information on the next card must begin from column 19.

Examples;

```
(1)  C    EXAMPLES OF KAIRO STATEMENTS
(2)       A1        AMP     100.0"A2
(3)       MAGTAP    OUT1    ABCD0038,2,1"RINP
```

```
(4)    ADDER     ADD     "BB1,BB2/
                          BB3,BB4

(5)    D         DEL     5"A1

(6)    FG        GEN     1,100,150,200,0.,123,0.,0."T1
```

Statement (1) is self-explanatory. Statement (2) expresses that block A1 is an anplifier with a gain of 100.0 which is connected with block A2 at its input terminal. It is shown in statement (3) that the output values of block RINP are filed with a file name of ABCD0038 on the magnetic tape device of logical number one. The positioning of the magnetic tape is directed by the second parameter, 2. In this case the corresponding operation is to record from a loading point and to rewind after recording. Block ADDER is an adder with four input blocks BB1, BB2, BB3 and BB4. The way to carry over the information of one statement to the next card is shown in statement (4). Block D is a delay line possessing five units delay. Block FG is a function generator producing the signal decided by its parameters at the time point (the relative clock time 1) when the control signal T1 becomes one, that is, logically true. It takes the values of 0.0, 123.0, 0.0 and 0.0 at the relative clock times 1, 00, 150 and 200 respectively, while it takes the interpolated values at nonspecified clock times, and the value at the last specified clock time is held by the time T1 becomes one again.

3-3-2 Statement for the change of parameters

The format of this statement is the same as that of the previous one. In the label field is written the label of the block whose parameter or parameters are desired to be changed. The function field has no

entry. The parameters are called first parameter, second parameter, etc., in the order in which they were listed in the statement specifying the block, and if the block has the integral parameters, including the alphabetic parameters in this case, and the real parameters, these are ordered separately. The pairs of the ordinal number and the value of the parameter which was to have been changed are listed, being separated by closed quotation-marks in the location corresponding to the parameter field and the input field. The order in which these pairs are listed may be independent of the ordinal number of a parameter.

Some parameters, for instance, the number of units delay of block type "DEL (delay line)", which are used to decide the structure of the corresponding object program cannot be altered by the above mentioned statement. The parameters which can be altered generally operate as mere constants. The statement to change the parameters will be exemplified below. The headed statement numbers are in correspondence with those of the previous examples.

(2)	A1	1,120
(3)	MAGTP	1,ABCD0039
(6)	FG	2,50"3,140"2,158.8

The gain of block A1 will be changed from 100.0 to 120.0 by statement (2). Statement (3) says that the next file name to be recorded on the magnetic tape device labeled MAGTP is ABCD0039. In block FG, the second and the third integral parameter will be changed from 100 to 50 and from 150 to 140 respectively, and the second real parameter from 123.0 to 158.8.

3-3-3 Modification of function by the omission of input blocks

The general aspects of KAIRO statement have been described in the previous sections. This section and the following two ones will be assigned to the description of the special but very characteristic aspects of it. One of them is that the functions of some block types can be modified by the omission of some inputs the positions of which are predetermined. It is an advantage for memorizing that the modification of the function has very intimate hardware correspondence. One such modification occurs in block type " BAT ". If it has one input, then it acts as a bias. On the contrary if its input is omitted, then it acts as a battery. Other example could be observed in block type " GEN " (see statement (6) exemplified in section 3-3-1). If it has no input, it generates the periodical signal whose period is indicated by its first integral parameter. These examples will be described by the following statements

```
B1      BAT    5.36E+01"A
B2      BAT    5.63E+01"
FG      GEN    200, 1, 100, 150, 200, 0., 123., 0., 0.".
```

3-3-4 Block type with multi-outputs

There are some block types which are able to offer more than two outputs in KAIRO language. This is one of characteristics of KAIRO language in comparison with other block diagram languages. Such block types contained in the current stockpile are so trivial that this facility does not seem so efficient. However it would have enormous potentiality in the future development of the language. For instance, assuming that the language would be developed so that it could accept

the subroutine written in KAIRO language, the fact that a block could offer multi-outputs would be very necessary, because the subroutine would often have more than two arguments whose values would be decided in the subroutines.

Each output of such block type must be labeled in order to distinguish one from the other, which is done with block type "DUM" as follows.

```
D       DIS    0.0"A1,C1
D1      DUM    "D
D2      DUM    "D
D3      DUM    "D
D4      DUM    "D
```

Block type "DIS" is a distributor with a signal input and a control input, which distributes the signal to the output terminal decided by the control input and the value of the parameter to the other output terminals. The order of the statements defining the labels of the outputs is so important that it decides the correspondence of a label to an output terminal.

3-3-5 Insertion of FORTRAN statement

It is almost impossible to design a language so that every facility which may be required in the future is provided. Consequently, it is desirable for the language to be made open-ended by allowing the programmer to design his own facilities in other languages. In KAIRO language it is permitted to write FORTRAN statements in a source program on the same level as the KAIRO statement. Special block type "FBOX" is provided for this purpose, which has no particular function, but is

put at the head of the sequence of FORTRAN statements to be inserted. A set of FORTRAN statements preceded by a "FBOX" statement acts as other block types provided in KAIRO language. With this block type the programmer is able to construct his desired functions or nonstandard input and output operations. He may write down not only a set of statements expressing his new function directly but also the calling sequence of the subroutine expressing it and add the subroutine itself to the generated object program in the execution run. One of the applications of this block type enables him to discover the occurrence of the abnormal situation of the model being simulated in the execution run and to dump the contents of core memory representing the state of the model or to cease the simulation. It should be noted that block type "FBOX" cannot define a new function code as the so-called programmer's macro can. If the new function defined with it is frequently used in a program, its definition, therefore, has to be written whenever it is needed.

Example 1

```
B       FBOX   2"A1,A2
     B=SIGN(1.5708(*),A1)          (*) 1.5708= π/2
     IF(A2.NE.0.0)     B=ATAN(A1,A2)
```

Example 2

```
A       FBOX    2
     READ(5,8000)  A1,A2,A3
8000 FORMAT(3F10.6)
A1      DUM     "A
A2      DUM     "A
```

```
A3          DUM     "A
```

A parameter of "FBOX" statement indicates the number of FORTRAN statements following it. The labels of blocks which are referred to in some FORTRAN statements must all be listed in the input field. The value decided with a set of FORTRAN statements is generally assigned to the label defined with "FBOX" statement as in example 1. If they decide more than two values, the labels for them must be defined with "DUM" statements as in example 2.

3-4 Control instructions and source program

3-4-1 Control instructions

Four control instructions, "KAIRO", "CHP", "END" and "FIN", are provided in order to define the program units constituting a source program of KAIRO language. "KAIRO" indicates that the sequence of the statements following it are those for the specification of a block. "CHP" indicates that the sequence of the statements following it are those for the change of parameters. "END", making a pair with "KAIRO" or "CHP", limits the range of influence of the paired control instruction. Furthermore "END" paired with "KAIRO" is able to appoint the clock time as an option at which the simulation will get through. "FIN" is placed at the end of the source program to make the language processor recognize that the source program is completed.

The control statement, as the statement including the control instruction code is called, is punched in the same format as that of a KAIRO statement. Instead of a blank, the symbol " # " is punched in column 1 in order to emphasize that the card contains the control

statement. The control instruction code is punched in columns 12 through 18 and the option starting in column 19 if necessary.

3-4-2 Source program of KAIRO language

There are two program units in KAIRO language. One of them describes the model to be simulated, and consist of the sequence of the KAIRO statements for the specification of a block stated in section 3-3-1, preceded by the control statement "KAIRO" and followed by "END". As a matter of course, it must contain the statements describing all the blocks in the model. The order of the statements contained may be independent of the constitution of the model, that is, the block diagram. Although the computation must be done through the signal flow, precisely according to the rules mentioned in section 3-2-3, the arrngement of the statements is left to the language processor. The programmer could arrange them in cases where the model was represented by a simply constructed block diagram, but it would be very difficult to do so in cases where the model was represented by a block diagram involving a number of blocks and very complex feed back loops. The automatic arrangement of the statements could lighten a programming burden considerably. The model to be simulated can be specified completely by this program unit.

There are three alternatives for specifying the required length of a simulation run. The first alternative is offered by the block types for data input such as "INP1" and "INP2". These block types can issue the command to stop the simulation spontaneously at the moment the data of the file just treated have been exhausted. The second method is to appoint the clock time to stop a current of the simulation with the option of an "END" statement. The details will be illustrated in

the next section. The third method is offered by block type "FBOX" as stated in the previous section.

Another program unit is employed to simulate the same model repeatedly while changing some of the parameters. It consists of the sequence of the statements for the change of parameters, headed by "CHP" and terminated by "END". The order of the statements contained may still be arbitrary. One program unit to alter the parameters corresponds to one of the repeated simulations. The parameters which have not been altered hold the old values. On the other hand, the parameters once changed hold the new values by the time they will be instructed again.

A source program in KAIRO language is defined with the program unit to describe the model only or this and some program units to change the parameters. In both cases it must be terminated finally by the control statement "FIN".

3-5 Example and its results

The response of a simple LC circuit shown in Fig. 3-3 to a sine wave was computed by KAIRO language, as a typical example. Let the input and the output of the circuit be x(t) and y(t) respectively, then the differential equation describing the circuit is as follows:

$$\frac{d^2y}{dt^2} + 2a\frac{dy}{dt} + (a^2 + w^2) y = (a^2 + w^2) x \qquad (1)$$

where $a = R/2L$ and $w^2 = 1/LC - R^2/4L^2$. The output response y(t) to the input $x(t) = E \sin(w_o t)$ was computed under the initial condition $y = 0$ and $\frac{dy}{dt} = 0$ at $t = 0$.

There are three approaches to solve the equation by menas of KAIRO language. The first approach is to compute the analytical solution of the differential equation (1) directly with FORTRAN. It can be obtained as follows after a primary computation,

$$y(t) = e^{-at} (A \cos(wt) + B \sin(wt)) + C \cos(w_o t) + D \sin(w_o t) \quad (2)$$

where $A = -C$

$$B = (aC + w_o D)/ w$$

$$C = -2aw_o(a^2 + w_o^2)E/\{ (a^2 + w^2 - w_o^2)^2 + 4a^2 w_o^2 \}$$

$$D = (a^2 + w_o^2)(a^2 + w^2 - w_o^2)E/ \{(a^2 + w^2 - w_o^2)^2 + 4a^2 w_o^2\}.$$

In the second approach, the representation of the equation (1) in terms of a Laplace transform is intermediate, as follows;

$$Y(s) = \frac{a^2 + w^2}{(s + a)^2 + w^2} X(s) \quad (3)$$

Therefore the circuit response in this case can be computed with block type "FLT2", which is implemented to compute the output response of the circuit represented in terms of a Laplace transform $(cs + d)/\{(s + a)^2 + b^2\}$. In the compiled form of block type "FLT2", the convolution integral of the input signal and the memory function of the circuit is approximated by the trapezoid rule.

The third approach is to construct a block diagram, referring to the corresponding representation of the equation (3) in terms of a z-transform. It is given by

$$Y(z) = \frac{z^{-1} e^{-aT} (a^2 + w^2) \sin(wT)}{w \{1 - 2z^{-1} e^{-aT} \cos(wT) + z^{-2} e^{-2aT}\}} X(z)$$

where T is the sampling period. The following two equations make it easy to transform the above equation into the block diagram;

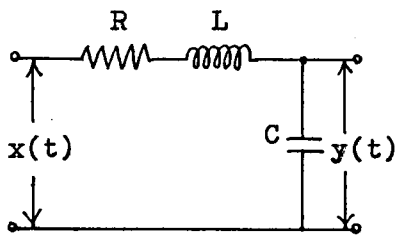


Fig. 3-3 The circuit for an example of KAIRO source program.

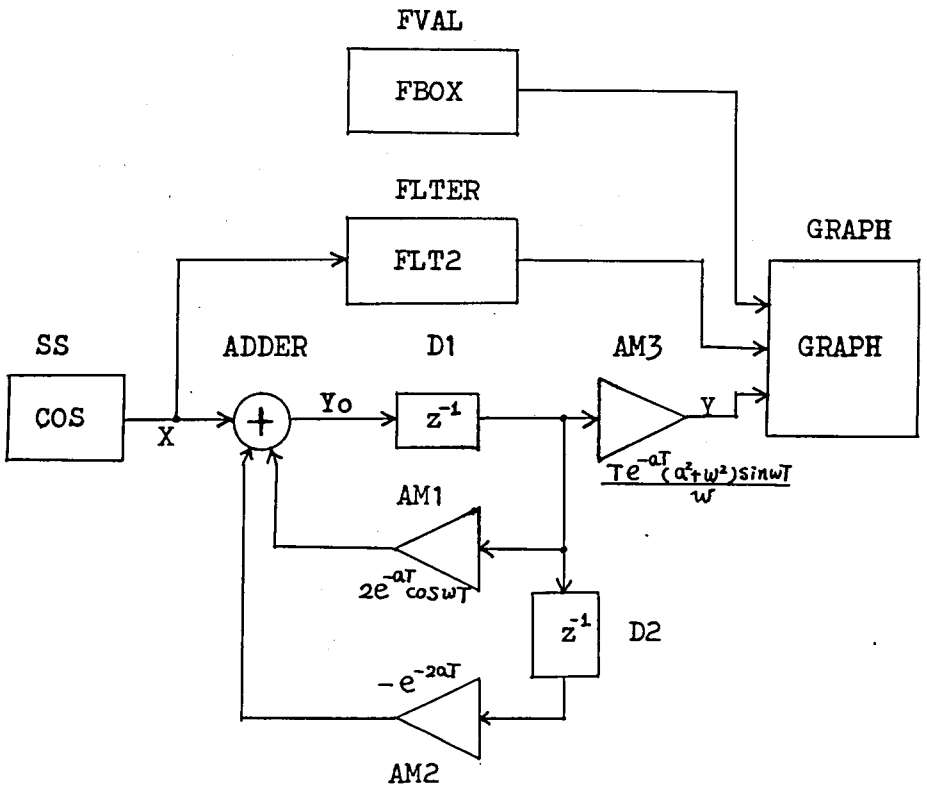


Fig. 3-4 The block diagram for computing the response of the circuit shown in Fig. 3-3 to a sine wave.

$$Y(z) = z^{-1}e^{-aT}(a^2 + w^2)\sin(wT)\, Y_o(z)/w$$

$$Y_o(z) = X(z) + 2z^{-1}e^{-aT}\cos(wT)\, Y_o(z) - z^{-2}e^{-2aT}\, Y_o(z). \qquad (4)$$

The three methods shown above are incorporated in Fig. 3-4. Block TVAL inserts the FORTRAN statements in the KAIRO program with block type "FBOX", which describes the procedure to compute the analytical solution (2) of the differential equation (1). Block FLTER computes the second output with block type "FLT2" according to equation (3). The blocks from block ADDER to block AM3 with the feed back loops composed of a delay line and an amplifier are constructed according to equations (4) and (5) to produce the third output. Block SS is a cosine generator to apply the sine wave $x(t) = E\sin(w_o t) = E\cos(w_o t - \frac{\pi}{2})$ to block FLTER and block ADDER. The three solutions are shown graphically by block type "GRAPH".

The block diagram appearing in Fig. 3-4 is specified by KAIRO language as in Fig. 3-5, in which the compiled lists generated by the KAIRO compiler are also included. It should be noted that block type "GRAPH" is compiled into a subroutine subprogram. The sampling period T, that is, the clock time interval with which the whole circuit operates, was 100 micro seconds. The amplitude E of the sine wave to be applied was 1000, and its frequency $w_o/2\pi$ was 500 cps, at first, which was changed to 300 cps with the CHP-program as shown in Fig. 3-5. The constants of the LC circuit, that is, the coefficients of equation (1) were set as follows,

$$a = 100\pi \text{ (rad/sec)}$$

$$w = 2000\pi \text{ (rad/sec) .}$$

Thus the coefficients appearing in the equations (3), (4) and (5) were

KAIRO SOURCE PROGRAM

```
#EXAMPLE   KAIRO  C

C EXAMPLE OF KAIRO PROGRAM
 SS         COS     1000.,500.,-1.5708,0.0001
 FLTER      FLT2    314.159,6283.18,0.,3.967721E+07,0.0001*SS
 ADDER      ADD     *SS,AM1,AM2
 AM1        AMP     1.568004*D1
 AM2        AMP     -0.9391048*D2
 D1         DEL     *ADDER
 D2         DEL     *D1
 AM3        AMP     0.3587910*D1
 TVAL       FBOX    28
      DIMENSION V(500)
      IF(IWATCH.NE.1) GO TO 8200
      READ(5,9000) F,E,F1,B,T
 9000 FORMAT(5F10.5)
      P=3.14159
      OM=2.*P*F
      OM1=2.*P*F1
      SG=P*B
      R=OM1*OM1+SG*SG
      S1=R-OM*OM
      S=S1*S1+4.*(OM*SG)**2
      SA=E*S1/S
      SB=-2.*SG*OM*E/S
      A=-SB
      B=-(SB*SG+SA*OM)/OM1
      XT=0.
      YT=0.
      OM=OM*T
      OM1=OM1*T
      P=2.*P
      EX=1.0
      EX0=EXP(-SG*T)
      DO 8100 I=1,500
      V(I)=(EX*(A*COS(XT)+B*SIN(XT))+SA*SIN(YT)+SB*COS(YT))*R
      XT=AMOD(XT+OM1,P)
      YT=AMOD(YT+OM,P)
 8100 EX=EX*EX0
 8200 TVAL=V(IWATCH)
 GRAPH      GRAPH   1,1,-1*TVAL,FLTER,AM3
#           END     WATCH,500
```

```
K003   GRAPH HAS NOT BEEN REFERRED.   WARNING ONLY
```

Fig. 3-5 Typical KAIRO program.

```
      KAIRO   COMPILED   LIST

*EXAMPLE   ID     HIJO8

*          HIJO8

*KK01      HARP

      SUBROUTINE PARA01
      COMMON IZ(0004),FZ(0013)
      IZ(0001)=0
      FZ(0001)=0.0
      FZ(0002)=1000.0
      FZ(0003)=500.0
      FZ(0004)=-1.5708
      FZ(0005)=0.0001
      FZ(0006)=314.159
      FZ(0007)=6283.18
      FZ(0008)=0.0
      FZ(0009)=3.967721E+07
      FZ(0010)=0.0001
      FZ(0011)=1.568004
      FZ(0012)=-0.9391048
      FZ(0013)=0.3587910
      IZ(0002)=1
      IZ(0003)=1
      IZ(0004)=-1
      RETURN
      END
```

Fig. 3-5 Typical KAIRO program (continued).

```
      KAIRO   COMPILED   LIST

*KKMAIN    HARP

      REAL SS,FLTER,ADDER,AM1,AM2,AM3,GRAPH
      COMMON IZ(0004),FZ(0013)
      DIMENSION MZ(0005),AZ(0013)
      MZ(1)=0
      AZ(1)=0.0
      ICNTRL=0
 5001 CALL CONTRL(ICNTRL)
      IF(ICNTRL.EQ.0) STOP
      IWATCH=0
      IEMGCY=0
      AZ(2)=6.283185*FZ(3)*FZ(5)
      AZ(3)=FZ(4)
      ZZ=EXP(-FZ(6)*FZ(10))
      AZ(4)=2.*COS(FZ(7)*FZ(10))*ZZ
      AZ(5)=ZZ*ZZ
      AZ(6)=-FZ(8)*AZ(5)
      AZ(7)=2.*(FZ(9)-FZ(6)*FZ(8))*SIN(FZ(7)*FZ(10))/FZ(7)
      AZ(8)=FZ(10)/2.
      DIMENSION GRAPH(100)
      MZ(2)=IZ(2)
      MZ(3)=IZ(2)+IZ(4)*IZ(3)
      MZ(4)=0
      MZ(5)=1
      D2=0
      D1=0
 5002 IF(IWATCH.EQ.00500) GO TO 5000
      IWATCH=IWATCH+1
      DIMENSION V(500)
      IF(IWATCH.NE.1) GO TO 8200
      READ(5,9000) F,E,F1,B,T
 9000 FORMAT(5F10.5)
      P=3.14159
      OM=2.*P*F
      OM1=2.*P*F1
      SG=P*B
      R=OM1*OM1+SG*SG
      S1=R-OM*OM
      S=S1*S1+4.*(OM*SG)**2
      SA=E*S1/S
      SB=-2.*SG*OM*E/S
      A=-SB
      B=-(SB*SG+SA*OM)/OM1
      XT=0.
      YT=0.
      OM=OM*T
      OM1=OM1*T
      P=2.*P
      EX=1.0
      EXU=EXP(-SG*T)
      DO 8100 I=1,500
      V(I)=(EX*(A*COS(XT)+B*SIN(XT))+SA*SIN(YT)+SB*COS(YT))*R
      XT=AMOD(XT+OM1,P)
      YT=AMOD(YT+OM,P)
 8100 EX=EX*EXU
 8200 TVAL=V(IWATCH)
      SS=FZ(2)*COS(AZ(3))
      AZ(3)=AMOD(AZ(2)+AZ(3),6.283185)
```

Fig. 3-5 Typical KAIRO program (continued).

```
      KAIRO   COMPILED   LIST

      ZZ=AZ(5)*AZ(9)
      AZ(9)=FLTER
      FLTER=AZ(4)*FLTER-ZZ+AZ(8)*(AZ(6)*AZ(11)+AZ(7)*AZ(10)+FZ(8)*SS)
      IF(IWATCH.EQ.1)FLTER=0.
      IF(IWATCH.EQ.2)FLTER=AZ(8)*((AZ(7)+FZ(8)*AZ(4))*AZ(10)/2.+FZ(8)*SS
     1)
      AZ(11)=AZ(10)
      AZ(10)=SS
      AM3=FZ(13)*D1
      IF(IWATCH.NE.MZ(2))GO TO 1002
      IF(MZ(2).EQ.MZ(3))GO TO 1002
      IF(MZ(4)+3.LE.100)GO TO 1003
 5100 CALL GRAPHZ(GRAPH,3,MZ(4),AZ(12),AZ(13),0,MZ(5),IEMGCY)
      IF(IEMGCY.EQ.1) GO TO 1004
      MZ(4)=0
 1003 JZ=MZ(4)
      JZ=JZ+1
      GRAPH(JZ)=TVAL
      JZ=JZ+1
      GRAPH(JZ)=FLTER
      JZ=JZ+1
      GRAPH(JZ)=AM3
      MZ(2)=MZ(2)+IZ(3)
      MZ(4)=JZ
 1002 CONTINUE
      AM2=FZ(12)*D2
      D2=D1
      AM1=FZ(11)*D1
      ADDER=SS+AM1+AM2
      D1=ADDER
      GO TO 5002
 5000 IEMGCY=1
      GO TO 5100
 1004 GO TO 5001
      END
```

Fig. 3-5 Typical KAIRO program (continued).

```
      KAIRO   COMPILED   LIST

*K6SUB     HARP
      SUBROUTINE GRAPHZ(A,N,JZ,XMAX,XMIN,NFP,IW,IE)
      DIMENSION A(100),M(101)
      DRUM DIMENSION D(10000)
      IF(IW.NE.1) GO TO 1
      NDO=1
      IW=0
      IF(NFP.NE.0) GO TO 3
      XMAX=A(1)
      XMIN=XMAX
      GO TO 2
    1 IF(NFP.NE.0) GO TO 3
    2 DO 100 I=1,JZ
      IF(XMAX.LT.A(I))XMAX=A(I)
      IF(XMIN.GT.A(I))XMIN=A(I)
  100 CONTINUE
    3 WRITE DRUM D(NDO),(A(I),I=1,JZ)
      NDO=NDO+JZ
      IF(NDO+JZ.GT.10000) GO TO 120
      IF(IE.NE.1) GO TO 200
  120 N2=100/N*N
      N3=1
      JQ=1
      STEP=0.1*(XMAX-XMIN)
      XM=XMIN+5.0*STEP
      WRITE(6,1000) STEP,XMIN,XM,XMAX
 1000 FORMAT(1H1///44X,33HGRAPHICAL OUTPUTS OF KAIRO SYSTEM//7X,5HSTEP=E
     110.3//7X,2(E10.3,40X),E10.3/11X,10(10H1---------),1H1)
      C0=10./STEP
      C1=-C0*XMIN+1.5
      NP=2**26
      NBL=10*NP
  130 IF(NDO-N3.LT.N2)N2=JZ
      READ DRUM D(N3), (A(I),I=1,N2)
      DO 10 I=1,N2,N
      DO 11 K=1,101
   11 M(K)=NBL
      WRITE(6,1100) JQ
 1100 FORMAT(1H ,3X,I5,2H +)
      DO 12 J=1,N
      K=I+J-1
      L=C0*A(K)+C1
      IF(L.GT.100) L=101
      IF(L.LE.0) L=1
      M(L)=(J-1)*NP
   12 WRITE(6,1200) M
 1200 FORMAT(1H+,10X,101A1)
   10 JQ=JQ+1
      N3=N3+N2
      IF(NDO.GT.N3) GO TO 130
      IW=1
  200 RETURN
      END
```

Fig. 3-5 Typical KAIRO program (continued).

```
          KAIRO   SOURCE   PROGRAM

#         CHP

 SS               2,300.
#         END
```

```
      KAIRO   COMPILED   LIST

*KK02       HARP

      SUBROUTINE PARA02
      COMMON IZ(0004),FZ(0013)
      FZ(0003)=300.0
      RETURN
      END
      FIN
```

```
      KAIRO   COMPILED   LIST

*KKCNTL     HARP

      SUBROUTINE CONTRL(I)
      I=I+1
      GO TO (01,02,03),I
 0001 CALL PARA01
      GO TO 100
 0002 CALL PARA02
      GO TO 100
 0003 I=0
 0100 RETURN
      END
*KKMAIN     ENTRY
*           EOF
```

```
    C008   PREFIX ID-CARD AND HIJOB-CARD AND SOME HARP DECKS IF ANY.
```

Fig. 3-5 Typical KAIRO program (continued).

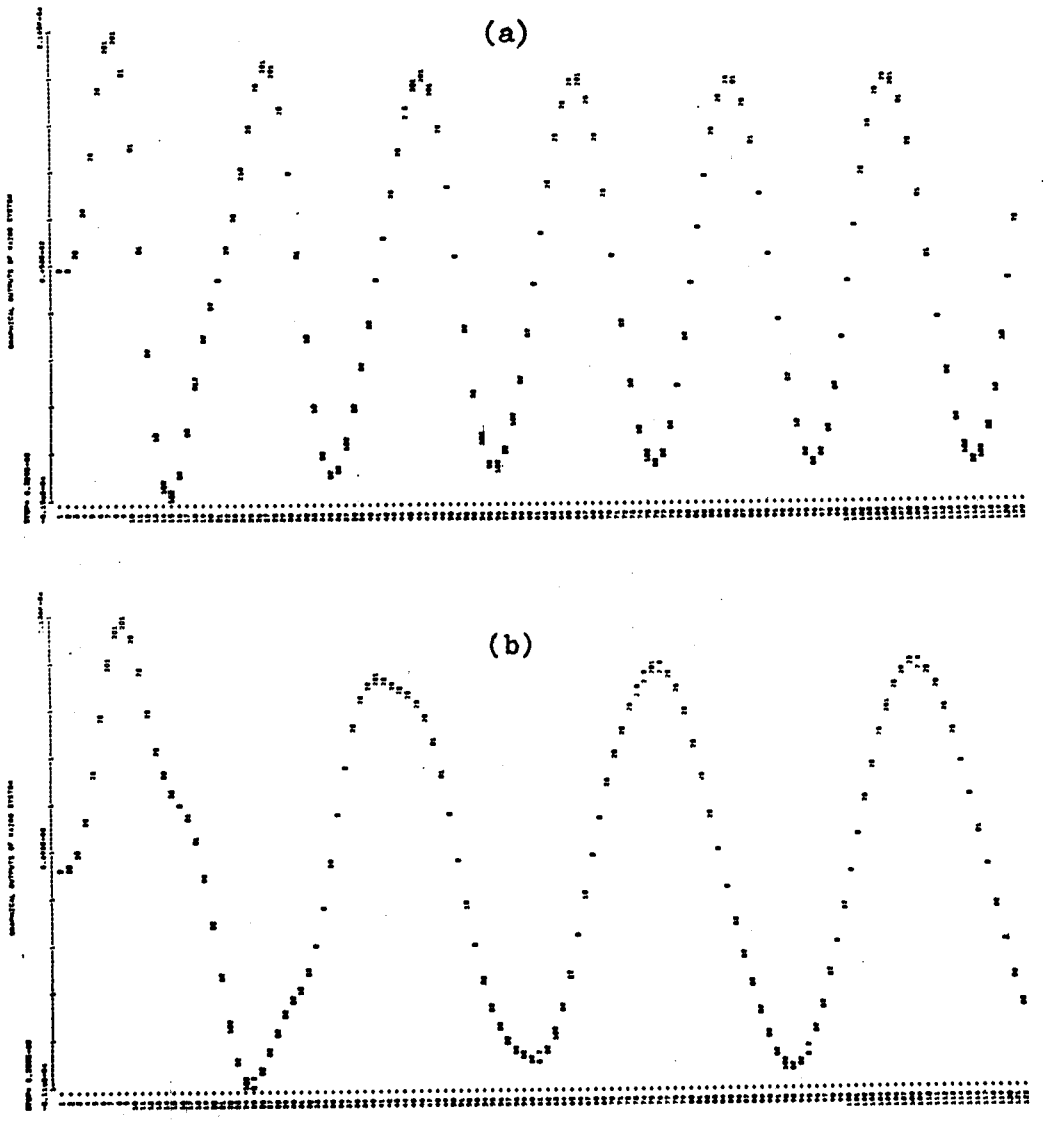


Fig. 3-6 Graphical outputs of results computed by KAIRO program illustrated in Fig. 3-5; (a) $w_o/2\pi$=500cps and (b) $w_o/2\pi$=300cps.

determined as follows,

$$a^2 + w^2 = 3.967721 \times 10^7$$

$$2e^{-aT}\cos(wT) = 1.568004$$

$$e^{-2aT} = 0.9391048$$

$$T\, e^{-aT}(a^2 + w^2)\, \sin(wT)/w = 0.358791$$

It should be noted that the factor T by which the output values of block SS was multiplied to approximate the delta-function was incorporated in the gain of block AM3.

It was stated that there were three methods for specifying the required length of a simulation run in section 3-4-2. In this example the second method was used. The clock time to stop the simulation run was appointed as the option of an "END" statement. In order to indicate the use of the option, "WATCH" was punched in columns 19 through 23 of that statement and digits to specify the clock time starting in column 24.

The results are shown graphically in Fig. 3-6. The symbols 0, 1 and 2 correspond to the first, the second and the third method respectively. The measure of agreement of the circuit responses by three different methods seems to prove them to be correct as supplementing one another. The output response is evaluated larger in absolute value by the trapezoid rule approximation of the convolusion integral and smaller in absolute value by the z-transform method.

3-6 Conclusion

The outline of KAIRO language, a simulation language which is best suited to the problems concerning information processing systems including both analog and digital signals, such as communication network

systems, speech analysis-synthesis systems and pattern recognition systems, has been described. KAIRO language is one of the block diagram languages. It enables the user to build up the model to be simulated in a hardware oriented form, that is,a block diagram, and so is used conveniently by engineers unfamiliar to the so-called general purpose languages.

It seems to increase the efficiency of the KAIRO language significantly that it allows the FORTRAN statements to be inserted on the same level as the KAIRO statement. The parameters of each block included in a block diagram, say, the model to be simulated can be altered conveniently by the statement for this purpose. Since the source program of KAIRO language is compiled in FORTRAN language, the two phases of the KAIRO compiler and the FORTRAN compiler must be passed through in order to simulate the model with KAIRO language. It will be inevitable that passing through these two phases will lower the efficiency of the final machine program. The efficiency of KAIRO language should be estimated on the basis of the total cost computed from the building up of the model to the obtaining of the simulated results. From this point of view it will be expected that a rapid conversion of the model into a simulation program more than makes up for its inefficiency.

Among the problems belonging to the same category are those for which a simulation language is best suited, or those for which it is not suited. Some models which can be represented with a block diagram can be simulated efficiently with KAIRO, but other models cannot. What kind of block diagrams is KAIRO unable to simulate efficiently? It is difficult to answer this question in general terms. However

it will be apparent that KAIRO is inadequate for the type of block diagram in which the necessary signal passes through one of the branches of the block diagram and the other branches are not in use. Although KAIRO is applied to a broad class of hardware oriented problems, it assumes that all the blocks included in the block diagram should operate efficiently in the whole process of the simulation. Therefore it does not possess the ability to recognize the idle parts of the circuits and to leave the computation of these parts undone. On the other hand, for the block diagrams which do not include the idle circuits, the object program of KAIRO may be expected to be as efficient in computer running time as a program directly written in FORTRAN can be.

If the programmer's macro and the subroutine composed of KAIRO statements could be available, KAIRO language would be extremely powerful. The programmer's macro facility and the subroutine facility are to define a new block type with a set of KAIRO statements and to give an abbreviated name to it. The user can call for it by the abbreviated name as if it were one of the predefined block types. They are nearly equal from the point of view of function, but are quite different in the complexity of compilation. This will be explained briefly in section 4-6. The small memory capacity of the available computer obstructs the implementation of these facilities.

CHAPTER 4

The KAIRO language processor

4-1 Introduction

A language processor can be named in various ways corresponding to the kinds of object languages and the methods of interpreting the source language. The KAIRO language processor should be called a preprocessor in the sense that its object language is not a machine language but FORTRAN, and a generator in the sense that the order of the statements describing a simulation model must not coincide with that of the corresponding portions in the execution of the object machine program. Disregarding these considerations, we shall call the KAIRO language processor a compiler which is a common name given to the language processor.

The following facts result from the fact that the object language of KAIRO is FORTRAN. The first is that the KAIRO compiler has only the responsibility for the translation of KAIRO into FORTRAN. The final machine program is executed under the supervisor or monitor in the operating system to which the FORTRAN compiler belongs. The second is that the procedure for each block type which must be prepared in the compiler is easier to write in FORTRAN than in the machine language. The third is that the quality of the object machine program can depend upon that of the available FORTRAN compiler. Its shortcoming is reflected at once in the quality of the object machine program. As to this point, however, it has been reported[28] that no remarkable difference between the procedure written in FORTRAN and the procedure written

in the assembly language exists in their efficiency, if the FORTRAN compiler has been preserved for a few years. In fact, it has been confirmed for some block types that the number of the machine instructions contained in the final machine program passing through two language processors is more by ten or twenty per cent than that in the program written directly in the assembly language by the author.

One of the most advantageous facilities of KAIRO language seems to be its facility for an automatic sequence control which contains the time advancing mechanism and the ordering mechanism of the blocks in the simulation model. In general, an advantage of a simulation language involves the complexity for the compiler. Since all the blocks in the simulation model have been assumed to operate synchronously in the present case (see section 3-2-3), the time advancing mechanism becomes so simple that the clock time can be advancing a unit at a time. The important problem for the KAIRO compiler to solve is the implementation of the automatic arrangement of the blocks in sequence through the signal flows.

4-2 Structure of the object program

The structure of the object program of KAIRO language appears in Fig. 4-1. It is the case that the model will be simulated (n+1) times while changing the parameters of the model. In Fig. 4-1, the variable J expresses the content of a clock counter, and the constant N expresses the required length of one simulation run. The constant N is equal to -1 unless it is appointed with an option in the control statement "END". In this case, the command to cease the simulation run must be issued from the block for this purpose, and so the control of the simulation

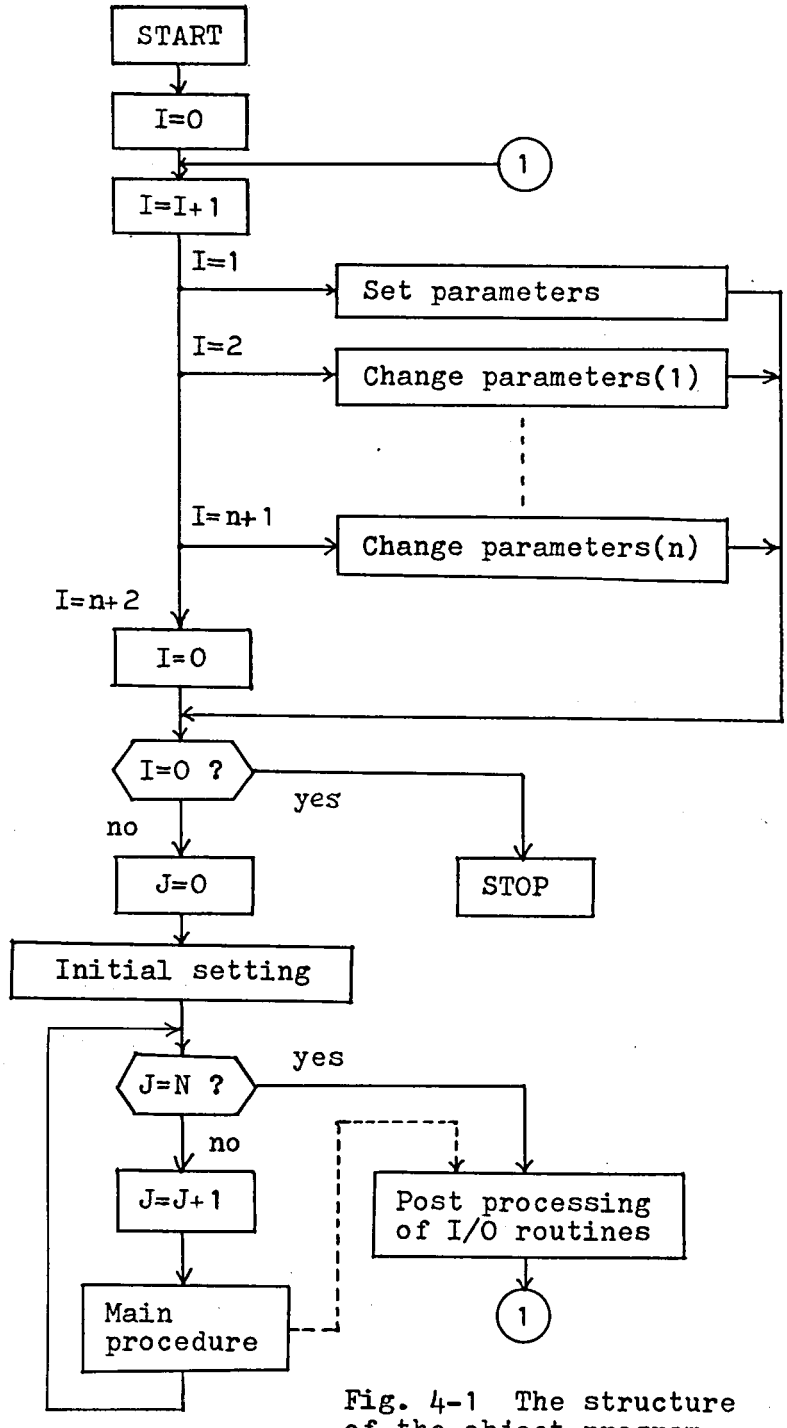


Fig. 4-1 The structure of the object program.

program will move along the path indicated by the broken line in Fig. 4-1. The program unit preceded by the control statement "KAIRO" is translated into the main program to describe the simulation model and the subroutine to set parameters. Each block type has the mould or original pattern expressed with FORTRAN-like notations in the compiler. It consists of a main operation part and an initial setting part if necessary. Corresponding to these parts of the mould, the main program describing the simulation model is also divided into the initial setting part and the main operation part.

There are two approaches in translating a block type. One is to generate directly the FORTRAN statements describing the corresponding procedure every time it is used. The other is to generate only the calling sequence for a closed subroutine describing the operation of the block type whenever it is used, and the closed subroutine itself only once, no matter how many times it may be called. The former takes a short computing time and a large space of core memory and fits those cases where the generated statements are rather few in number. On the other hand, the latter takes a small space of core memory and a long computing time due to the overhead time of the closed subroutine, and fits those cases where the number of statements involved in the subroutine is so many that the overhead time can be neglected. Most of the problems to be investigated with KAIRO language require a number of repeated computations at one simulation experiment, for example, 20,000 repetitions to process the speech data of a second sampled at the sampling period of 50 micro seconds. The former was adopted, therefore, for most of the block types in KAIRO language so that the computing time could become as short as possible. Some block types, however, were compiled into

the closed subroutines in order to spare the core memory and to simplify the structure of the object program.

With KAIRO language, it is permitted to repeat the simulations while changing a part of the parameters included in the model. In order to implement this facility, the values of the parameters were not embedded in FORTRAN statements as constants, but were set in the array assigned for the parameters of each type. The constants embedded in FORTRAN statements could not be changed by means of FORTRAN statements. The array names "IZ" for the integral and the alphabetic parameter and "FZ" for the real one were used for this purpose. As to setting the parameters, the available FORTRAN compiler had the shortcoming during the period when the KAIRO compiler was being developed that the statement "BLOCK DATA" could not be used[27]. If this statement were available, the arithmetic expressions, for example, FZ(30) = 43.3, would not have to be used in order to set them in the arrays and the computing time and the core memory could be spared.

The program unit preceded by the control statement "CHP" is translated into the subroutine to change the values of the parameters required. One "CHP" program unit corresponds to one subroutine, which consists of the arithmetic expressions as exemplified above. Finally when the control statement "FIN" is read in and the number of changes of the parameters is decided, the subroutine to control the linkage between the main program and one of those subroutines is generated. The structure of the object program, that the procedure for this linkage is not embedded in the main program but constituted as a subroutine subprogram, has been determined on the principle of decreasing the drum memory space for the generated program as much as possible.

4-3 KAIRO compiler

4-3-1 The outline of the KAIRO compiler

The outline of the KAIRO compiler will be described in this section. The outlined flow chart of the compiler appears in Fig. 4-2. Corresponding to each block type, the table which contains the information necessary to analyze the statement describing the block in the block diagram being simulated, and the mould expressed with FORTRAN-like notation, which is the origianl pattern used to generate the FORTRAN statements describing the function of the block type, are prepared in the compiler. The table, called "Function table (F-table for short)", consists of eight words per block type and involves the following information:

(1) Function code.

(2) Some indicators representing the special properties of the block type such as a delay line, a block with no inputs, etc..

(3) the number of each kind of parameters.

(4) the number of each type of inputs.

(5) information on the modification of the function by the omission of inputs.

(6) the number of outputs and their signal types.

(7) the origin address of the mould.

(8) the origin address of the special processing program if necessary.

The detailed description of the original pattern will be given in later section.

The main part of the compiler is divided into the following four

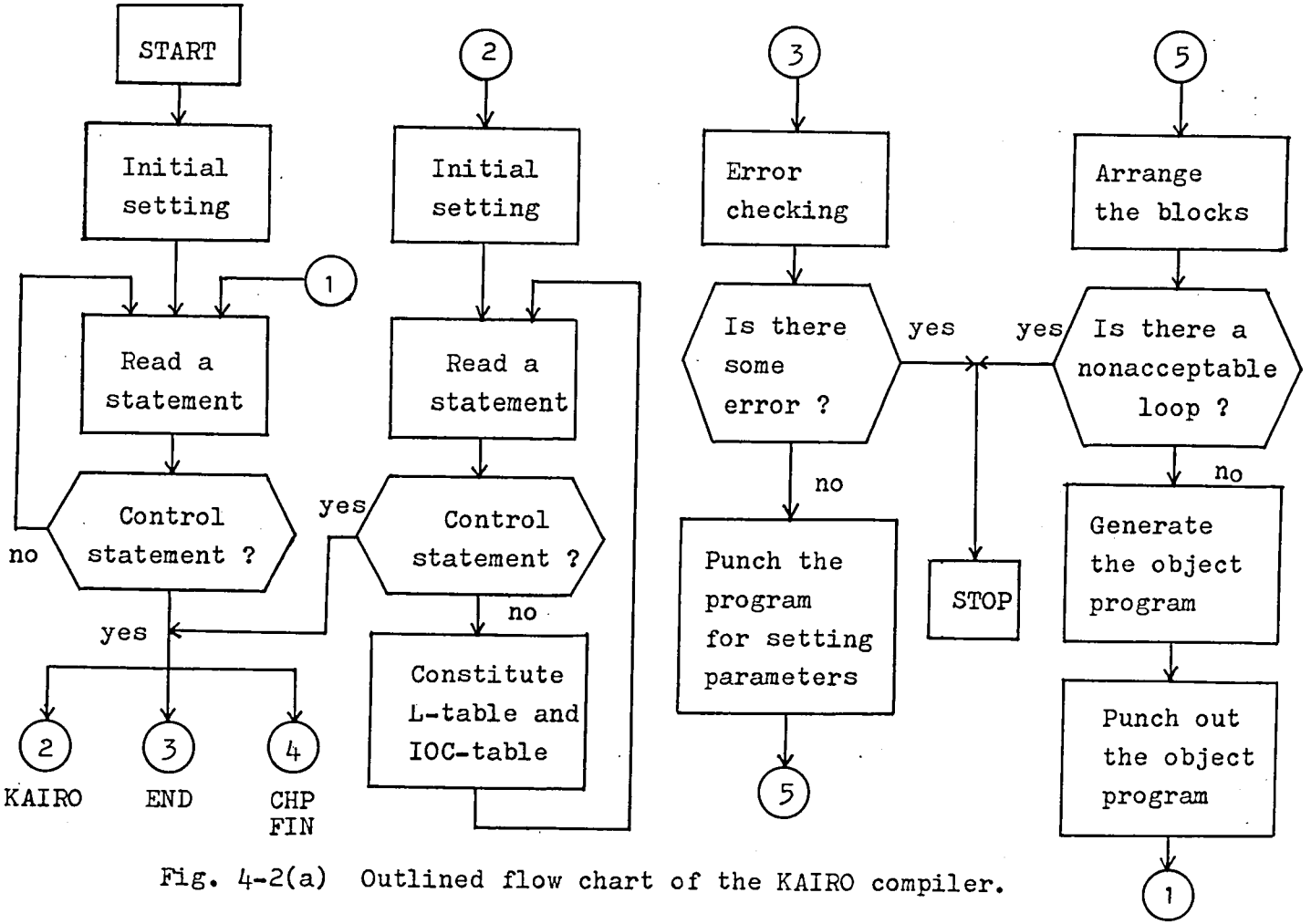


Fig. 4-2(a) Outlined flow chart of the KAIRO compiler.

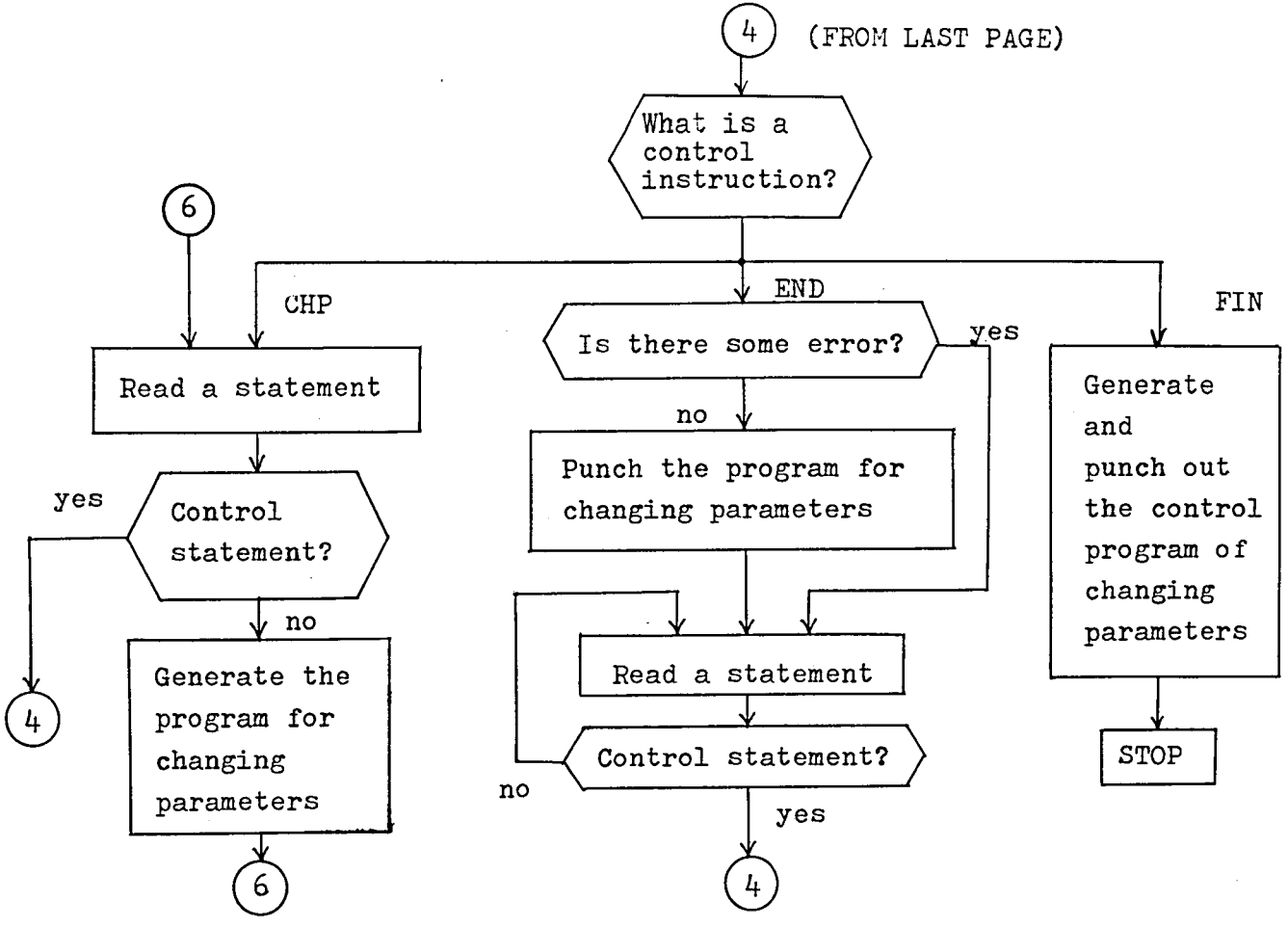


Fig. 4-2(b) Outlined flow chart of the KAIRO compiler(continued).

phases.

(1) The input phase reads the source program describing the block diagram specifications and generates a table named Label table or L-table for short in preparation of later phases. The input and output connection of the blocks is also stored in the form of bi-directional list structure. This is named the IOC-table for short.

(2) The arrangement phase, using the IOC-table, arranges the blocks in sequence through which the simulation of the blocks will be able to be executed correctly in the computer.

(3) The generation phase generates the FORTRAN statements corresponding to each block and puts them out on the appropriate output device.

(4) The parameter change phase translates the programs preceded by the control statement "CHP" into the subroutine subprograms altering the parameters and also generates the subroutine subprogram controlling the repetitions of simulation runs.

The KAIRO compiler is written in the assembly language of HITAC 5020, and consists of about 8,700 words including the mould of each block type and is divided into ten chain links, some of which are in correspondence to each phase mentioned above. The available computer, HITAC 5020, has a core memory of 16K words. The system monitor and input/output system subroutines called by the KAIRO compiler occupy about 9,700 words. The memory assignment of the compiler in the available memory space is as follows. The core resident part of the compiler consists of the main program controlling the chain links, several fundamental and frequently called subroutines and the F-table, occupying about 1,200 words. Each chain link occupies at most 1,000 words, and 4,500 words remain as working space, which is able to accept a block

diagram including about 300 blocks.

4-3-2 Input phase

The input phase reads the statements specifying the block diagram and generates the L-table and the IOC-table in preparation for later phases. The L-table consists of seven words for each block and includes the following information:

(1) the label assigned to the block by the programmer.

(2) the number of characters of the label.

(3) the output signal type copied from the F-table.

(4) the number of input blocks.

(5) the number of blocks fed to the output.

(6) the numbers of the integral and the real parameters.

(7) the values of the subscripts in the parameter arrays assigned to the first parameters of both kinds.

(8) the link addresses of the input and the output connection.

(9) the origin address of the F-table of the block type used in this statement.

(10) other temporary information required, such as some indicators explained later.

In the IOC-table, the input and the output connections are stored in the form of the bi-directional list structure. As soon as the statement specifying a block is read in, the information on the input connection can be determined, but that on the output connection can not. It is, therefore, advantageous for saving the memory space that the information on the output connection is also stored in the form of the list structure. The L-table and the IOC-table are shown in Fig. 4-3(b)

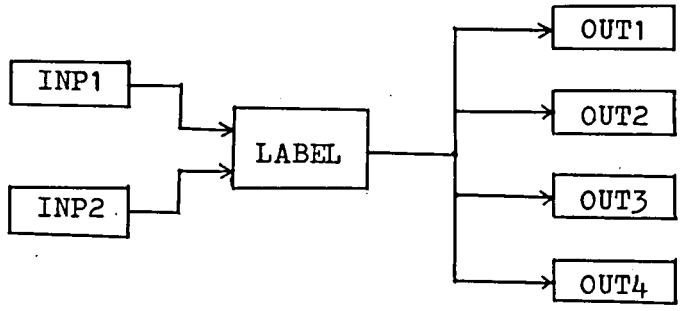


Fig. 4-3 (a) An example of the input and output connection.

(1) LABEL	
(10)	(6)
(4)	(5)
(8) INORG	(7)
(8) OUTORG	LINK2
(6)	(9)
(2) and (3)	(7)

← Origin address of the L-table of the block named " LABEL "

Fig. 4-3 (b) The L-table. INORG is the link address of the input list and OUTORG that of the output list. Other columns contain the information explained with the corresponding numbers in the text.

OUTORG

OUT1	LINK1

LINK1

OUT2	LINK2

LINK2

OUT3	

← temporary end mark

INORG

INP1	
INP2	end mark

Fig. 4-3 (c) The IOC-table. In the present case, block OUT4 is assumed not to be specified yet and so it does not appear in the IOC-table.

and (c), together with an example of the input and output connection of a block in Fig. 4-3(a).

The flow chart of the input phase appears in Fig. 4-4. The input phase consists of the following six subparts.

(1) Subpart I checks for the format of the label and looks up its entry in the L-table. If the label has been found there, it is ascertained whether it is multi-defined or not. The subsequent processing of that statement will be deleted, if it is so. If it has no entry in the L-table, the locations for it are secured there. Instead of the label, a string of characters, the origin address of these locations will be used to represent the name of the block in later processes. The indicator denoted by the character "U" in Fig. 4-4, representing that the function has been defined, is set on.

(2) Subpart II examines first whether the function code coincides with one of those provided in the F-table or not. The origin address of the F-table belonging to that block type is written down in the L-table, if it is acceptable. This origin address will be substituted for the function code itself to represent the block type in later phases. There are some block types that must be subjected to special treatment. For instance, FORTRAN statements following the "FBOX" statement are read in here and stored in the drum area for this purpose. The number of statements to be read in is determined by the parameter of the "FBOX" statement. Such a parameter is read here as the number of units deley of block type "DEL", which does not operate simply as a constant, but determines the structure of the compiled program.

(3) Subpart III checks for the formats of the parameters and generates the statements to set the parameters in the arrays for this purpose

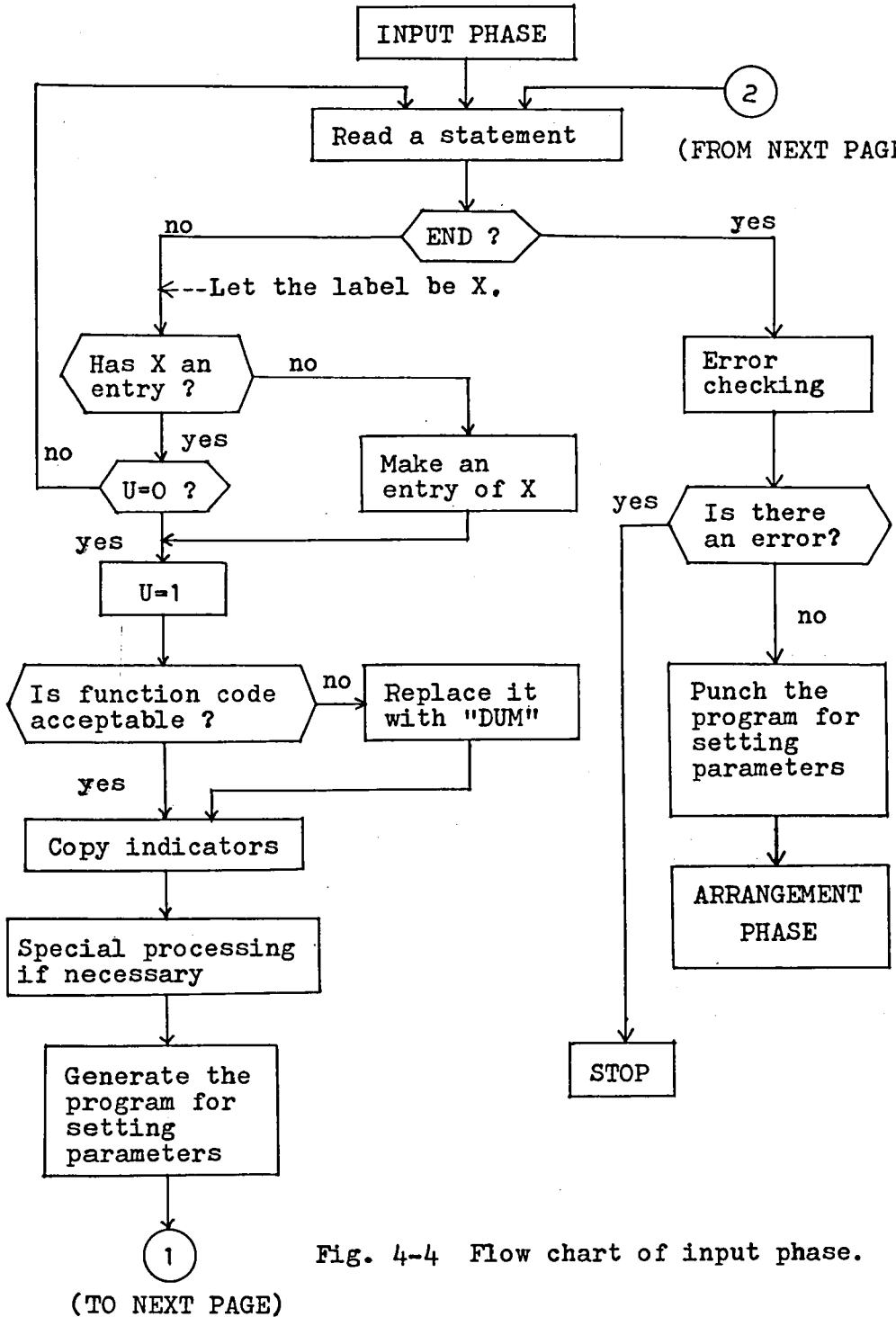


Fig. 4-4 Flow chart of input phase.

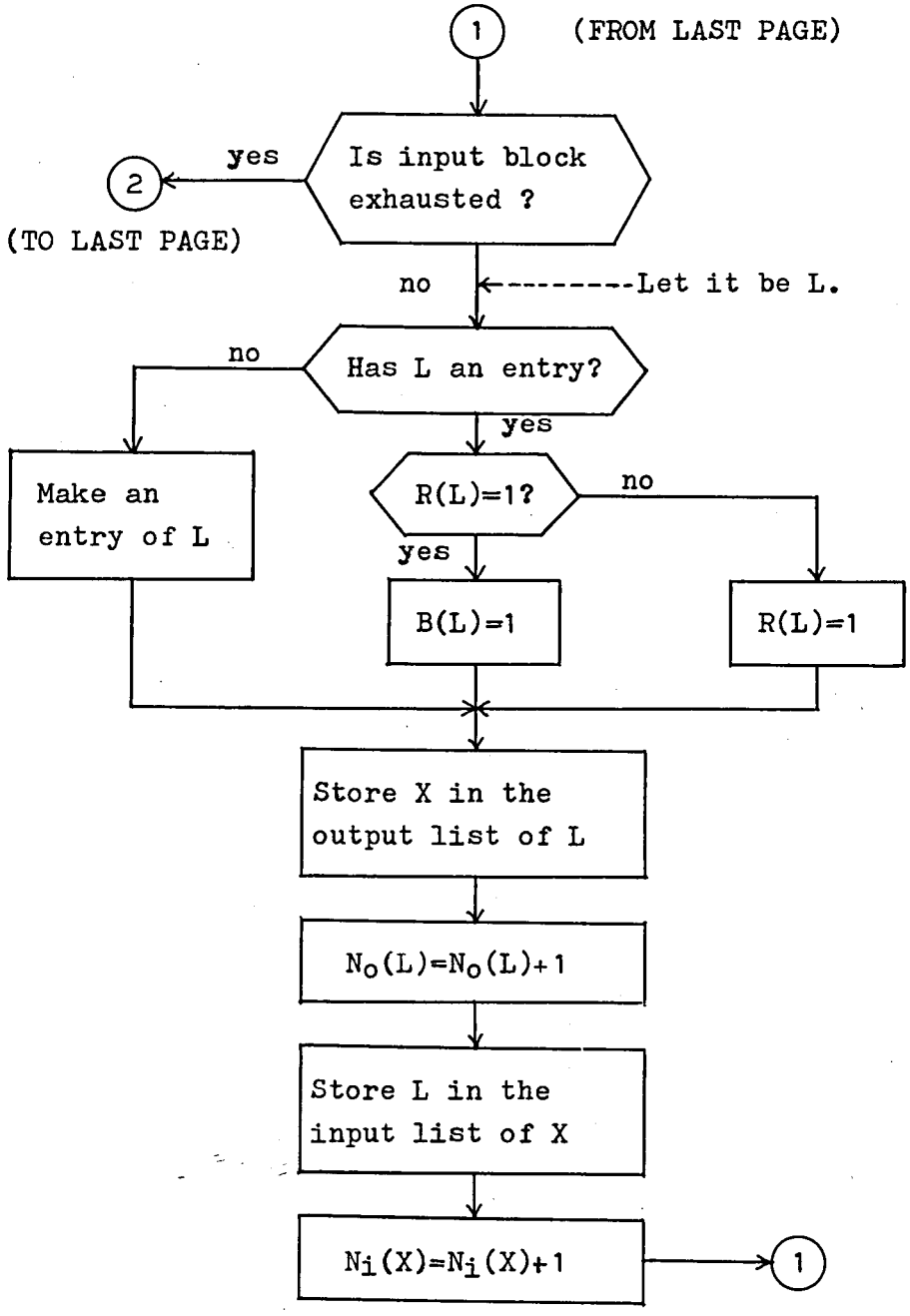


Fig. 4-4 Flow chart of input phase(continued).

and stores them in the drum area. The array names "IZ" for the integral parameter and "FZ" for the real one are used. Provided that the content of the subscript counter corresponding to the first, N1, of the integral parameters is thirty, the statement is in the following form and one is added to the subscript counter;

IZ(30) = N1.

Similarly for the second and third integral parameter, N2 and N3, the statements,

IZ(31) = N2

and

IZ(32) = N3

are generated. The value of the subscript assigned to the first of the integral parameters and that of the real ones are recorded in the L-table and finally the numbers of the integral and the real parameters are checked with those written in the F-table respectively.

(4) Subpart IV looks up the entries of the labels appearing in the input field consecutively in the L-table, and makes an entry of the label which has not yet been recorded. For the block which was recorded in the L-table, the indicator, denoted by R(L) in Fig. 4-4, indicating that the output of the block is connected with an input of another block, is tested. If it is not already set on, this is done. Otherwise, the indicator, denoted by B(L) in Fig. 4-4, indicating that the block appears in the input fields more than twice, is set on. The number, $N_o(L)$, of the blocks fed to the output of the block being treated is increased by one, and a branch is further added to those of the list towards the output. For the block just defined, the number, $N_i(X)$, of the input blocks and the link address of the input connection are written down in

the L-table. In the locations following the link address of the list towards the input, the names of the input blocks are recorded consecutively in the form of the origin address of the L-table. Provided that some of the input blocks are omitted, the modification of the function is done after it is insured that no grammatical violation of the input omission exists.

(5) To subpart V, the control is transferred when the "END" statement is read. Here the existence of the undefined label, which appears in the input fields of some statements but not in the label field, is examined and the matching is done in the number of the input blocks and their types for each block recorded in the L-table. If any rule violations are discovered in steps (1), (2), (3), (4) and (5), the consecutive compilation is abandoned at the time when the process of subpart V is completed.

(6) Subpart VI reads from the drum the FORTRAN statements to set the parameters, generated in subpart III, and puts them out to the appropriate output device in the form of a subroutine subprogram.

4-3-3 Arrangement phase

This phase arranges the blocks in sequence through the signal flow by using the bi-directional list structure constituted in the input phase. The algorithm for arrangement is closely related to the representation of a delaying-type block in the computer. The policy of the KAIRO compiler is that only one word memory is assigned for one unit delay. Furthermore, all the contents of a delay line are to be initialized zero. Since this is a special operation different from that of a delay line at other clock times, it may be rather convenient to control

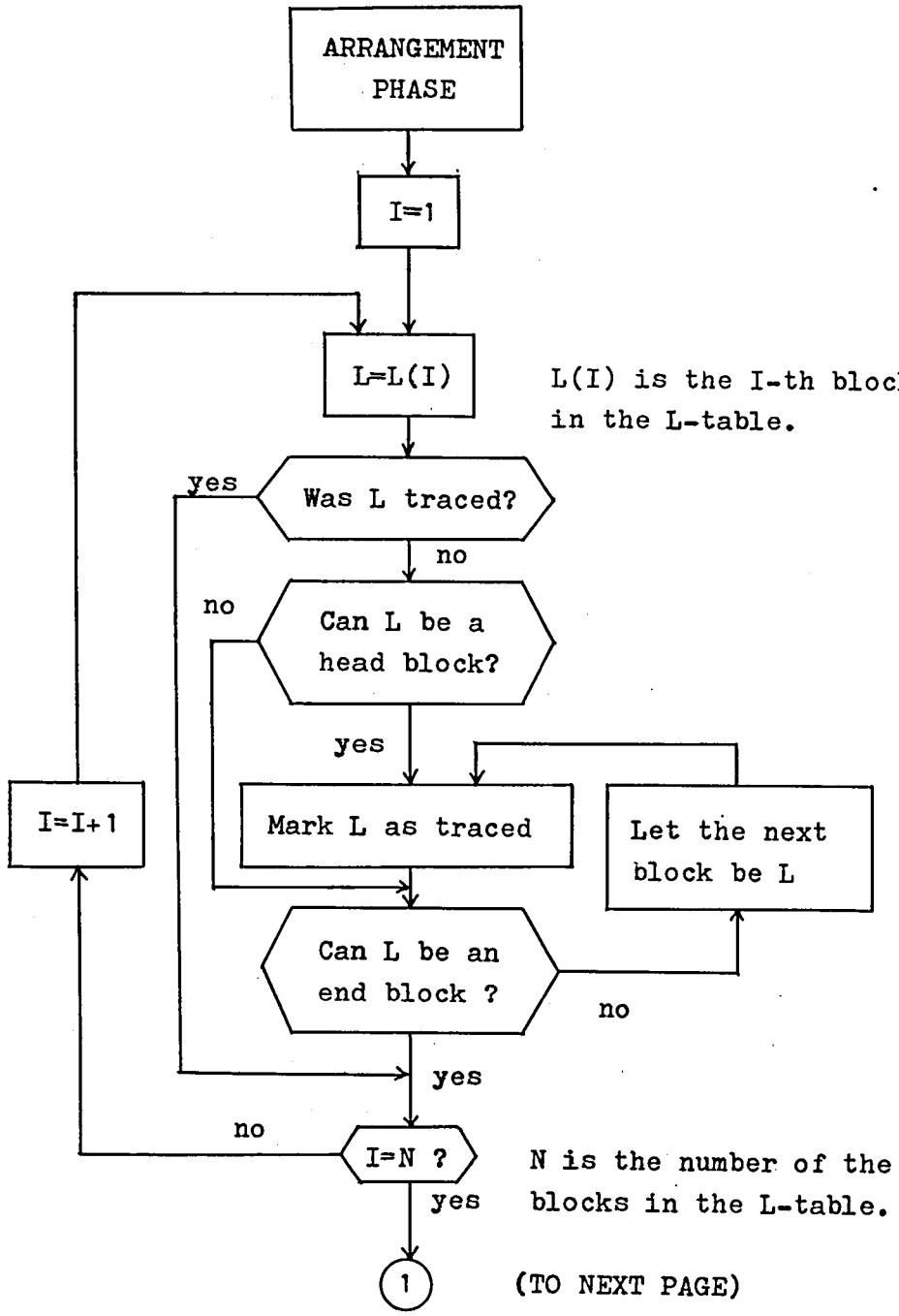


Fig. 4-5 Flow chart of arrangement phase.

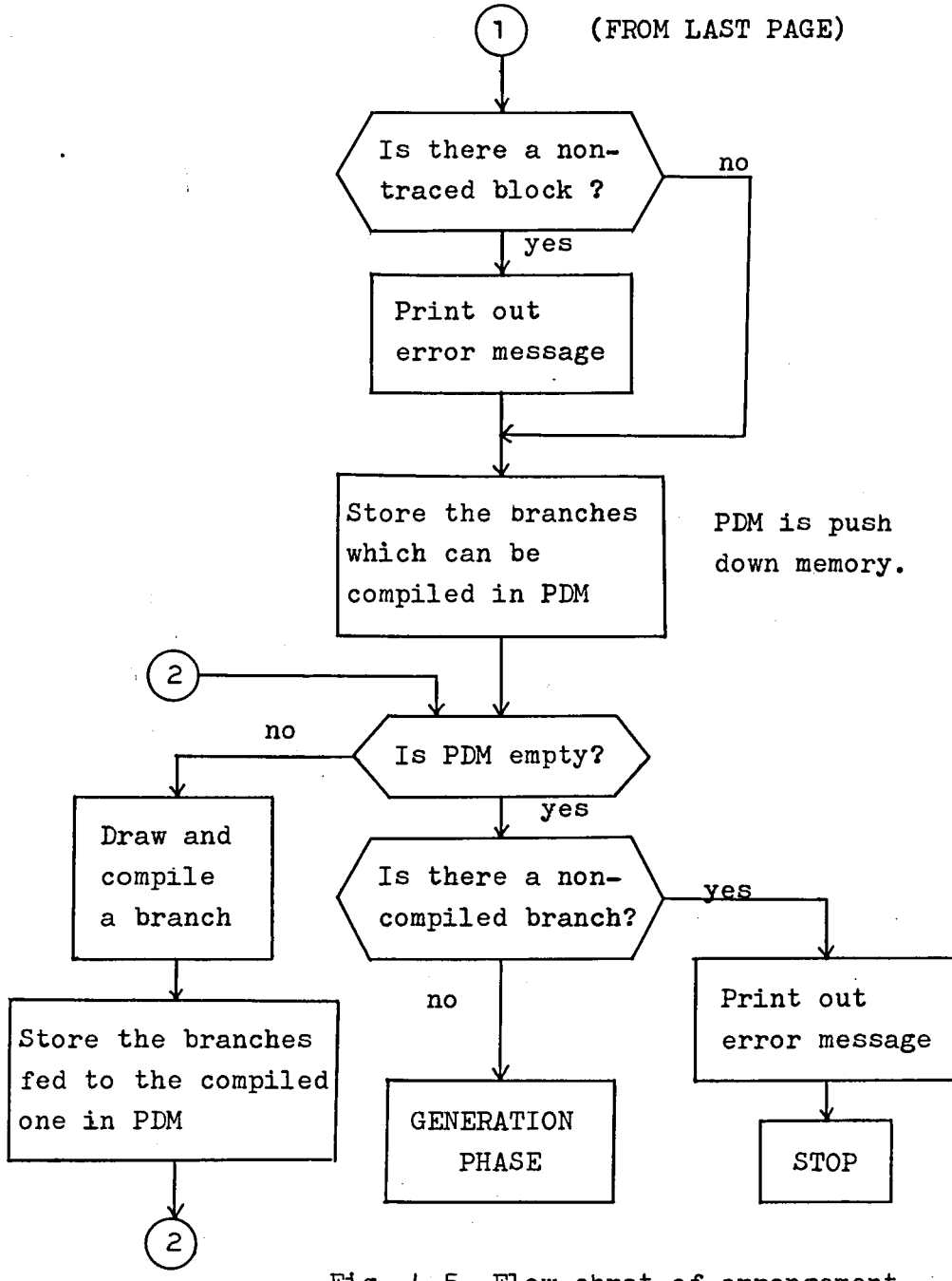


Fig. 4-5 Flow chrat of arrangement phase(continued).

the object program so that, at the present clock time of the simulation, the current output can be computed for a nondelaying-type block, while the output for the next clock time can be computed for a delaying-type block. Thus the alogorithm for arrangement can be derived from the discussion in the section 3-2-3 as follows:

(1) a nondelaying-type block cannot be computed before all the input blocks to it except delaying-type blocks have been computed.

(2) a delaying-type block cannot be computed before all the input blocks except delaying-type blocks and all the blocks fed to it have been computed.

The flow chart of the arrangement phase is shown in Fig. 4-5. This phase consists of the following two part.

(1) Search of a branch. A branch is defined as a sequance of successively computable blocks. At first the arrangement phase searches a branch for the purpose of computing the blocks included in it consecutively. The block which can be the head block of a branch is a block with no input such as a generator, a block with more than two inputs, a block with a delaying-type block as input, or a block which has a single input and whose input block is fed to more than two blocks. The block which can be the end block of the branch is a block being fed to the head block of another branch or a block, such as a printer, which is not fed to any other blocks. These blocks are shown in Fig. 4-6 (a) and (b). The block which can be the head one and simultaneously the end one forms a branch by itself. A delay line is forced to be a branch by itself for simplicity of programming. All the blocks appearing in the block diagram should be included in some branch. The block which was not included in any branch must be included in a

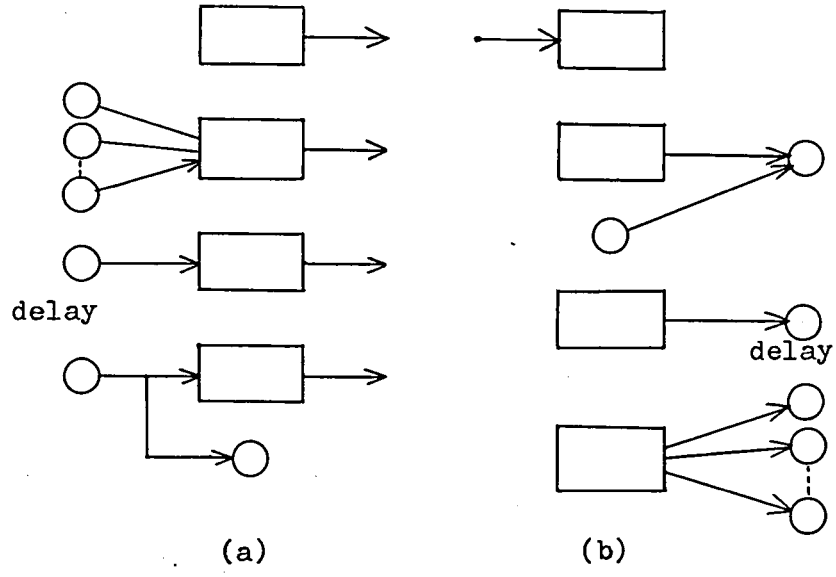


Fig. 4-6 The blocks which can be the head or end block: (a) head blocks, (b) end blocks.

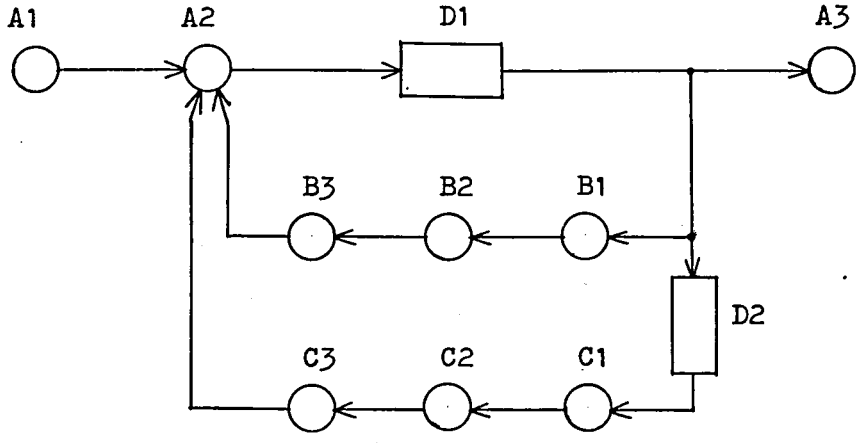


Fig. 4-7 A block diagram for explaining the arragement phase; blocks represented by rectangules are delay lines and blocks by circles are nondelaying-type blocks.

closed loop consisting of nondelaying blocks only. In Fig. 4-7, blocks B1, B2 and B3 and blocks C1, C2 and C3 form branches. The first branch will be denoted as B and the second as C for simplicity. Blocks A1, A2, A3, D1 and D2 form branches by themselves respectively. First the head block is searched out and the bi-directional list structure is traced towards the output from the head block, marking the blocks as traced, until the end block can be found. This process is continued until all the blocks have been traced.

(2) Arrangement of branches. Since the connection of the blocks can be replaced with that of the branches by procedure (1), the branches should be arranged in sequence through the signal flow as the next step. The first computable branch is that whose head block has no input or has only delaying-type blocks as inputs. The branches satisfying either of these requirements are all stored in push-down pop-up memory. The branch is, then, drawn from the push-down memory according to the rule "last in - first out", and compiled. The word "compiled" means here that the "compiled" branch is signed in order to assume that all the blocks included in that branch have been computed. For the branches fed to the branch just compiled, it is tested in turn according to the algorithm for arrangement to see whether they become computable, and those which become so are stored in the push-down memory. The same procedure is then repeated, modifying the list structure towards the output to express the sequence of arranging the blocks, until the push-down memory has no entry and so the arrangement is completed. The branches that have not been compiled in this procedure are turned to be included in the loop consisting of delaying-type blocks only or nondelaying-type blocks only.

	CONTENTS			
(1)	A3	A1	B	C
(2)	A3	A1	B	D2
(3)	A3	A1	B	
(4)	A3	A1		
(5)	A3	A2		
(6)	A3			
(7)	D1			
(8)	EMPTY			

Fig. 4-8 The transition of the contents of the push-down memory.

Fig. 4-8 shows the transition of the contents in push-down memory during the procedure for the block diagram appearing in Fig. 4-7. The branch in the extreme right hand column is drawn out first. Now assume that the contents of row (1) are as shown in Fig. 4-8, although they might depend upon the order of the statements describing the block diagram shown in Fig. 4-7. Branch C is compiled first and then branch D2 becomes computable. It is stored in the push-down memory and at once drawn from it and compiled. Branches D2, B and A1 are compiled in turn and then branch A2 becomes computable. Last of all branch D1 becomes computable after branches A2 and A3 have been compiled. The computation order of the blocks are, thus, decided as follows; C1, C2, C3, D2, B1, B2, B3, A1, A2, A3 and D1.

4-3-4 Generation phase

This phase generates the FORTRAN statements corresponding to the blocks according to the order determined in the previous phase. The compiled forms of the block types in KAIRO, of course, have different structures from each other. So it is not convenient to provide the corresponding procedure to generate the FORTRAN statements for each block type. In order to treat most block types by a rather simple program, the original pattern called a statement mould is prepared for each block type. It is a sequence of statements with notations similar to FORTRAN statements and including some operators.

The compiler scans the symbols in the statement mould sysmbol by symbol. Meeting an operator, and making reference to the information contained in the L-table and the IOC-table, the compiler interprets it as will be mentioned below. The nonoperator symbols are incorporated, as they are, into the corresponding FORTRAN statement. When the symbol " & ", which indicates the end of one statement in the mould, is found, the FORTRAN statement just generated is transferred to the buffer memory for this purpose. After the compilation of all the blocks included in the block diagram is completed, the generated statements are put out to an appropriate output device together with some additional type statements and specification statements. If the block which should be compiled into a subroutine subprogram is included, it is put out thereafter. The flow chart of this phase is shown in Fig. 4-9.

There are three kinds of basic operators used in the statement mould.

(1) The replacement operator takes the form of "#AB(C)" and is replaced by some information determined by the KAIRO statement describing

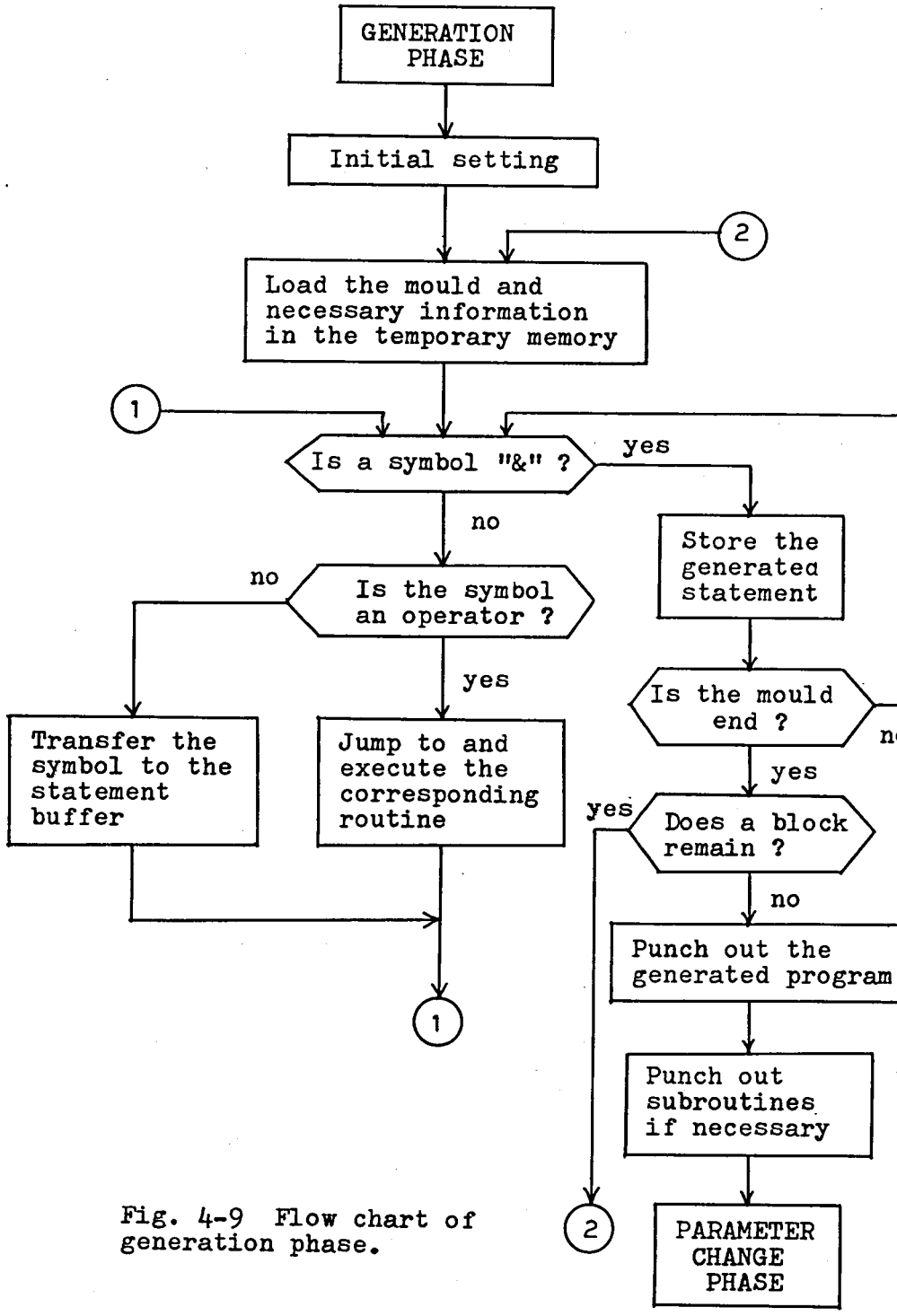


Fig. 4-9 Flow chart of generation phase.

a block. The symbol "#" indicates that two or three symbols following it are to be interpreted as an operator. The symbol "A", expressing what kind of information the operator is replaced by, is one of the symbols "L", "I", "P", "F", "M", "A" or "S", representing a label, an input block, an integral parameter, a real parameter, a working variable of the integral type, that of the real type and a statement number respectively. The information expressed by the symbol "A" is specified in detail by the next symbol "B", which is one of the symbols "X" and "N" and a digit. The symbol "X" means that the term represented by the preceding symbol is specified by the content of the repetition counter, explained in reference to the repetition operator, which plays the same role as a decimal number does as will be mentioned below. The symbol "N" means that the operator is replaced by the number of the terms expressed by the symbol "A", for instance, the number of the input blocks. If the symbol "B" is a digit, then the third symbol "C" is also interpreted as a part of the operator. Provided that a decimal number "BC" is "03" and the symbol "B" is "I", the operator "I03" is replaced by the third input label.

(2) The repetition operator is employed to express simply the undetermined part of the statement mould such as the number of the input blocks of an adder, and takes the following two forms; "#AR" and "#A". There is a counter called the repetition counter to control the repeated interpretations of the part expressed by the repetition operator. In its first form, the second symbol "A" is a variable which has the same meaning as in the replacement operator, and "R" is a constant symbol indicating that the operator is the repetition one. This operator sets the number, corresponding to "#AN" in the replacement operator, to the

upper limit of the repetition counter. The symbol "A" of the second form is one of the symbols "+", "-" and """. The "+" forces the addition of one to the content of the repetition counter and the "-" the subtraction of one from it. The pair of the strings "#"" denotes the range of the undetermined part. The compiler, meeting the first "#"", initializes the content of the repetition counter, and, finding the second "#"", examines whether or not it coincides with the upper limit of the counter. If they are not equal, one is added to the content of the counter and then the flow of the compiler program is controlled to interpret the part following the first "#"" again. Otherwise, the repeated interpretations have been completed. The nesting of the pairs of the strings "#"" is not allowed.

(3) The deletion operator is used to delete some statements or a part of a statement in the mould conditionally. This operator takes the form of "#¥A.....#¥". The symbol "A" has the same meaning as in the replacement operator. If the number involved by "#AN" is zero, the part enclosed by the pair of the strings "#¥" is deleted.

Example

The shift register appearing in Fig. 4-10 is described by the following KAIRO statements.

```
A       SHR    "B,T
A1      DUM    "A
A2      DUM    "A
A3      DUM    "A
```

The statement mould of the block type "SHR" is as follows;
for the initially setting part,

```
#LR#" @ #LX=0&
#"
```

and for the main part,

```
 @ IF(#I02.EQ.0) GO TO #S01&
#LR#+#"#- @ #LX=#+#LX&
#"
 @ #LX=#I01&
 #S01 CONTINUE&
```

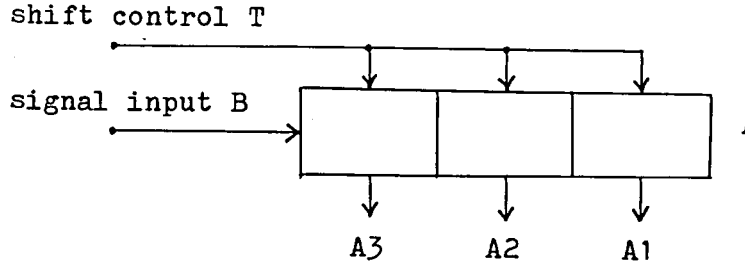


Fig. 4-10 A shift register with three cells.

The symbol " @ " is substituted for four blanks. In the present case, the number to be set to the upper limit of the repetition counter by the operator "#LR" is equal to three and the operators "#I01" and "#I02" are replaced by the labels "B" and "T" respectively. The operator "#LX" is interpreted as "A1", "A2" or "A3", depending upon the current content of the counter. Assuming that statement number "1009" is assigned to the operator "#S01", the compiler interprets the statement mould and generates the following FORTRAN statements corresponding to the block type "SHR". For the initial setting part,

```
      A1 = 0
      A2 = 0
      A3 = 0
```

and for the main part,

```
      IF(T.EQ.0) GO TO 1009
      A1 = A2
      A2 = A3
      A3 = B
 1009 CONTINUE.
```

4-3-5 Parameter change phase

This phase reads the program units preceding the control statement "CHP" and generates the subroutine subprograms altering the parameters. One "CHP" program unit corresponds to one subroutine subprogram. The procedure of this phase is very similar to that of subpart III in the input phase. It checks the formats of the parameters and examines whether or not the ordinal number of the parameter to be changed can be accepted. For instance, assuming, for the block labeled "A", that the number of the integral parameters was three and that of the real ones was two, and that the value of the subscript in the array for the integral parameter was thirty and that for the real one was forty, the statement instructing the change of the values of the parameters,

```
A                    1,30.0"2,4.0"1,17
```

can be accepted and translated into the arithmetic statements in FORTRAN in the following form.

```
      FZ(40) = 30.0
      FZ(41) =  4.0
```

IZ(30) = 17

However, the statement,

A 1,3.0"3,5.8

cannot be accepted because the number of real parameters must not be greater than two. For the statements including any rule violations, the compilation is omitted and the error messages are issued.

This phase reads the control statement "FIN" too and generates the subroutine subprogram to control over the flow of the object program so that the simulation runs for the block diagram being studied are repeated the same numbers of times as the number of the correct "CHP" program units previously counted. Thus, the whole process of compiling a source program of KAIRO language has been completed. The flow chart of this phase is omitted because of its simplicity.

4-3-6 Programmer's macro and subroutine

A programmer's macro is a new funtion which can be defined by the programmer with the combination of the block types prepared in KAIRO language, and can be employed in the same way as the predefined block types. A subroutine is also defined with the sequence of KAIRO statements as the programmer's macro, and can be called by the new function code. Their programming advantage lies in the fact that the statements to define the process required are only presented once in the program, no matter how many calls for them are made by the main program. They can contribute to omitting the burden of writing the same patterned program sequence repeatedly and to decreasing the possibility of making careless mistakes. Although the current form of the compiler cannot be impremented for these facilities, a brief account of the algorithm

to perform them will be given in this section.

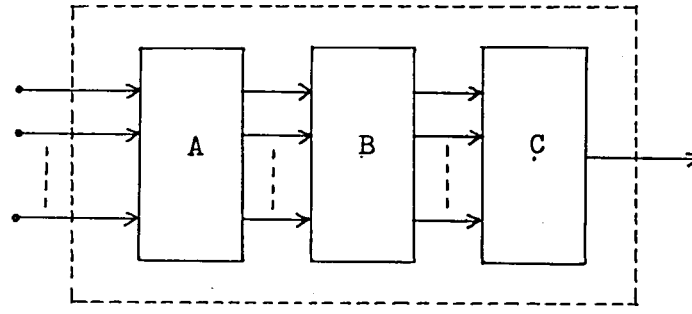
In the interpretation of the programmer's macro, the statement defining the macro code is read first, and it is supplemented in the F-table in preparation for its use in the main program. As the compiler is reading the statements defining the function, the L-table and the IOC-table for the macro definition, which contain the dummy variables, such as input labels and parameters to be replaced with the actual ones determined in its use, will be constituted in the same way as is done in the input phase of the compilation of the "KAIRO" program unit. Finding the statement including the macro code in the "KAIRO" program unit, the compiler will transfer all the contents of the L-table and IOC-table for the macro definition into the corresponding tables provided for the "KAIRO" program unit, replacing the dummy variables only used to define it with the actual ones just decided. This results in the compiler reading the statements obtained by replacing the dummy variables with the actual ones. Thus, the implementation of the programmer's macro seems to be rather simple.

The subroutine of KAIRO language is different from that of a genearl purpose language in that the outputs of it, say, the arguments whose values can be computed in it, happen to be decided independently of the present inputs, say, the arguments whose values have already been decided before calling it; in other words, in that they can have delays. The possibility that some outputs have delays and that others have no delays will greatly increase the difficulty.

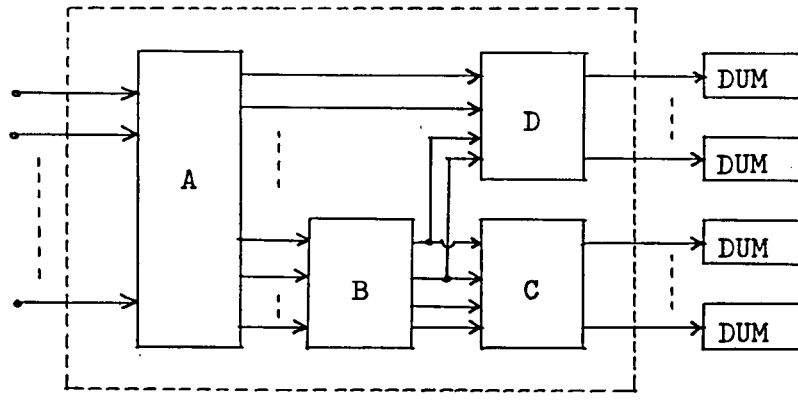
Now it is assumed that the subroutine has a single output. If it can be computable when the blocks contained in the subroutine are arranged in sequence through the signal flow on the assumption that no

inputs to the subroutine have been determined, it can be decided to have delays. Otherwise, it must be insured that it has no delays so that it can be computable when the blocks are arranged on the assumption that all the inputs have been determined, because, if the subroutine has any error, it can happen that it will not be computable on either of the two assumptions stated above.

The nondelaying-type subroutines written in KAIRO language can easily be translated into the corresponding ones written in FORTRAN by almost the same procedure as that used to translate the "KAIRO" program unit. On the other hand, the delaying-type subroutine is not so simple as a delay line. In general, the delaying-type subroutine can be shown schematically in Fig. 4-11(a). Although its output can be computed independently of the present values of the inputs, only the blocks included in the part labeled "C" in Fig. 4-11(a), can then be computed. The other blocks cannot be computed before its inputs have been determined. If its output is fed into one of its inputs through other nondelaying-type blocks, all the blocks defining the subroutine cannot be computed successively. This makes it difficult to take the subroutine as one block, when the blocks included in the main program, that is, the "KAIRO" program unit, are arranged in sequence through the signal flow. This difficulty can be overcome by constituting the compiled program so that the initial value may be computed by the use of the part labeled "C" in Fig. 4-11(a) before entering the main routine and all the blocks in the subroutine may be computed to produce the output at the next clock time in the main routine as done in the simple delay line. Of course, the two calling sequences must be provided for the call for computing the initial value and the call for the main routine.



(a)



(b)

Fig. 4-11 The general structure of a delaying-type subroutine; (a) has a single output and (b) has multi-outputs.
In both figures, part A is composed of nondelaying-type blocks and/or delaying-type blocks, part B delaying-type blocks only, part C and part D in (b) nondelaying-type blocks only.

Finally we will turn to the case in which the subroutine has more than two outputs. Their types can be decided by the two arrangement procedures previously stated. In the case that some outputs have delays and the others no delays, the type of the subroutine itself cannot be decided. This difficulty can also be overcome by translating the delaying-type outputs in the way as just mentioned and assigning their types to the output labels defined by the "DUM" statements as shown in Fig. 4-11(b). Thus, the body of the subroutine can be taken as a non-delaying-type.

4-4 Error checking and error message

One of the complexities accompanying the simulation program is the control of the sequence in which the events or the phenomena occur in the model being studied. The investigator is not of great value in this connection and tends to make a program carelessly. However, program errors made here often produce incredible results and are extremely difficult to discover and to eradicate. It has been stated in chapter two that a decided advantage of a simulation language compared to a general purpose language is the ability to check for not only grammatical violations, but also for some logical errors. With KAIRO language, the programmer need not be explicitly conscious of a problem of sequence control, which can be achieved spontaneously when the blocks can be arranged in sequence through the signal flow on the basis of the connections of the blocks. The sequence control error corresponds to the possibility of arranging the blocks in the signal flowed sequence from the point of view of error checking. This possibility of arrangement

is examined in the arrangement phase of the compiler. Most kinds of errors which are not introduced in converting the model into the program, but are included in the model itself are higher order problems and, of course, cannot be discovered by the compiler.

The compiler checks for the following rule violations mainly due to the programmer's careless mistakes: the usage of invalid characters in a label assigned to a block, the format error of parameters, the usages of the function codes undefined in KAIRO language, the multi-defined labels and the undefined labels, and the mismatching in the number of parameters, the number of input blocks and the types of input blocks. The overflows of the tables such as the L-table and the IOC-table are examined. An error message is printed out, as a rule, at the time when the error is found. Examples of the error messages issued by the compiler are shown in Fig. 4-12.

KAIRO SOURCE PROGRAM

```
#EXAMPLE   KAIRO
C  EXAMPLES OF ERROR MESSAGES
 AD         ADD     "AM3,AM4, ,F

      X    T010    SOME ERROR OCCURS ABOUT BLANK INPUT.

 AM1        AMP     1.38"AD
 AM2        AMP     3"AM1

      X    T007    NUMBER OF PARAMETERS IS INVALID.


      X    T007    NUMBER OF PARAMETERS IS INVALID.

 D1         DEL     2"AM2
 D2         DEL     1"D1
 AM3        ANP     5.8"D2

      X    T005    F-PART IS UNDEFINED AND REPLACED BY "DUM".


      X    T007    NUMBER OF PARAMETERS IS INVALID.

 AM3        AMP     -30.6"D1

      Y    T004    ABOVE LABEL IS DOUBLY DEFINED.   THE STATEMENT IS SKIPPED.

 SCHMT      SCH     -0.3,-0.6"AM2
 TRGER      TRGN    "SCHMT
 CNTER      CNT     0,10,9"SCHMIT, ,TRGER
#           END
```

```
Y   K002   AM4    HAS BEEN UNDEFINED IN SOURCE PROGRAM.

Y   K002   F      HAS BEEN UNDEFINED IN SOURCE PROGRAM.

Y   K002   SCHMI HAS BEEN UNDEFINED IN SOURCE PROGRAM.

    K003   CNTER HAS NOT BEEN REFERRED.   WARNING ONLY

SINCE THERE IS SOME ERROR IN SOURCE PROGRAM, COMPILATION CANNOT BE CONTINUED.
```

Fig. 4-12 Typical error messages issued by the KAIRO compiler.

CHAPTER 5

The intelligibility of infinitely clipped, time quantized speech wave —— an application of KAIRO language to speech information processing (I).

5-1 Introduction

The applications of KAIRO language to speech information processing were reported in this and next chapter. For the purpose of harmonizing these chapters with the previous ones devoted to the description of KAIRO language, an applied aspect of KAIRO language was rather accentuated than a detailed description of the problems to be studied. Although these problems were described only in their essential parts, it may be beyond question that these are the important problems worth to be investigated in fields of speech information processing.

The properties of speech wave concerned with the time dimension were investigated in this chapter. The intelligibility of simplified speech wave in the amplitude-dimension and the time dimension was measured and by analyzing these results, the confusion matrices of various phonemes were constructed. The simplification involves dichotomization of the amplitude of speech wave and quantization of the time length of an interval between two adjacent zero-crossing points.

The present experiment is one of the suitable applications of KAIRO language, because it is easy to perform it rather with computer simulation than with physical devices. Before the description of the experiment, general properties of speech wave and the necessity for its investigations will be stated briefly.

5-1-1 Genearl properties of speech wave

We find the following in the opening sentence of Fant's work.[29] "The speech wave is the response of the vacal tract filter systems to one or more sound sources." This expresses vividly the common view of those who devote themselves to studies of speech wave in engineering. It means mathematically that the characteristic P(s) of the observed speech wave can be expressed in terms of the Laplace transform;

$$P(s) = S(s)\ T(s)\ R(s)$$

where S(s) is the characteristic of a sound source, T(s) is the characteristic of the vocal tract filter systems and R(s) is the characteristic of the radiation from a mouth. The radiation R(s) can be taken as constant independently of sounds as first approximation, and the sound source S(s) can be an aperiodic source produced by the vibration of vocal cords and a noise source generated by the constriction or the closure of the vocal tract. The transfer function T(s) of the vocal tract filter systems is closely related to the shape of the vocal tract and is the mathematical representation of an acoustic resonance and anti-resonance in the mouth cavity system including the nose cavity. The frequency characteristic of T(s) gives such a great variety to speech wave that it can be used for an indespensable means of human communication as well as characters. Above all the spectral peaks of the spectrum T(f) which are called formants correspond closely to the phonemic representation of speech sounds. These formants are named first formant, second formant, etc. in order they occur in the frequency scale.

Information on the naturality due to speaker's emotions and individuality as well as linguistic information for human communication

are conveyed by the speech wave. It is well-known that speech wave has considerable redundancy from the point of view of the information theory, regarding it only as a means of transmission of linguistic information. However it may be said that communication is the more possible in very noisy circumstances and we can recognize the better a particular speaker in several ones because speech wave has such an amount of redundancy.

If we attempt to transmit the linguistically significant information of speech wave in telephone channel, we may establish efficient transmission system by eliminating or decreasing of its redundancey. In general, these methods are referred to as bandwidth-compression systems.

As understood easily from the process of speech production, the redundancy of speech wave is due to the continuity of its wave form and the mutual dependency of temporally successive values, which corresponds to the multi-dimensional Markov process in the discrete case. Although this fact would suggest the possibility of an efficient encoding process according to the coding theory of Shannon-Fano, the direct application of the coding theory to the case of speech wave would require the extremely complicated encoding-decoding process and a great amount of storage for a long time delay. Therefore, the actual bandwidth-compression systems are rather the attempts to extract and transmit the cues conveying the linguistically significant information by analyses of speech wave and to synthesize it by the use of them at the destination than the attempts to decrease its redundancy by the direct application of the coding theory. In this sense, such bandwidth-compression systems are generally referred to as analysis-synthesis systems.[30] The comprehensive knowledge of physical characteristics of speech production, perception and language is necessary as a background of construction of these systems.

5-1-2 Zero-crossing wave

Zero-crossing wave of speech sounds, which is obtained simply by clipping of the peaks of the speech wave until it is reduced to a two-valued function of time, seems to offer the clue for the bandwidth compression. The classic experiments of Licklider and Pollack[31] demonstrated that the infinitely clipped speech wave had higher intelligibility, although its amplitude was so distorted. It was useful to improve the intelligibility that infinite clipping was preceded by differentiation and was followed by integration. Surprisingly the articulation score of such wave which was measured by the use of monosyllabic words was rather 95 per cent.

The mathematical structure of zero-crossing wave was studied in terms of an analytic signal of the original speech wave.[32] However, the question why zero-crossing speech wave has such high intelligibility has not been answered sufficiently because it is closely related to the perceptual mechanism of the auditory system of human beings. It was only shown experimentally that the nonlinear transformation of infinite clipping preserved the formant structure in the frequency spectrum of the original speech wave fairly well.[33][34]

Since infinite clipping dichotomizes the amplitude dimension and no further reduction can be achieved in amplitude, it is necessary to operate upon the temporal patterns if the speech wave is to be further simplified. Such an experiment was carried out by Licklider.[35] In order to make a difference between his experiment and the present one, the former will be described in detail somewhat.

In his experiment the simplifying operation was quantization of the time scale. In quantized time a rectangular wave can switch only at

predetermined instants which are set by a train of pulses that divides the time scale into intervals. Two methods of time quantization were tested. According to method A, the output wave switches at the end of an interval if, during the interval, the input wave has switched one or more times. According to method B, the output wave switches at the end of an interval if, during the interval, the input wave has switched an odd number of times. With quantizing rate between 2,000 and 10,000 quanta per second, method B yielded higher intelligibility score than did method A in articulation tests conducted for subjecting speech wave to such a treatment. With 10,000 or more quanta per second, intelligibility was approximately as high as it was with RC filtering and infinite clipping alone (95 per cent).

The effects upon confusions within and between phoneme categories of subjecting speech wave to infinite clipping have been reported most recently.[36] Vowellike sounds such as vowels and semi-vowels are less often confused than voiced and unvoiced fricativelike ones. Stop and nasal consonants are intermediate between these.

5-2 Plan and procedure of experiment

5-2-1 Plan

Referring to the experiments on infinitely clipped speech wave described in the previous section, the present experiment was planned with the aims:

(1) to study intelligibility of subjecting speech wave to infinite clipping and quantizing of the time length of an zero-crossing interval, and

(2) to construct the confusion matrices of various phonemes transformed in such a way.

The temporal patterns of zero-crossing of infinitely clipped wave is the same as that of original speech wave. In the present experiment further reduction was performed on the temporal patterns so that the time length of an interval between two adjacent zero-crossing points was transformed into one of intervals which have predetermined widths. This transformation is quantization of the time length of a zero-crossing interval which is a continuous quantity. In Licklider's experiment[35] also referred to as time quantization, it was time scale that was quantized. Thus, as shown in Fig. 5-1, an zero crossing interval whose time length is shorter than a quantum of time scale vanishes and the probability that several intervals are incorporated into one interval increases as the quantum is longer. On the contrary, it was rather the time length of a zero-crossing interval than time scale that was quantized in the present case and the interval, no matter how short it is, of course at the extent of sampling period in computer simulation, can be transformed into one interval without vanishing. Thus, total time length of speech wave transformed in such a way may increase or decrease depending upon methods of quantizing, though the number of zero-crossing intervals does not change. For this reason, computer simulation seems to be rather adequate than physical devices which are complicated considerably do.

The used methods of quantizing were linear in terms of frequency scale (in the range of less than 1500 cps). The width of an interval between two adjacent zero crossing points was measured by counting a train of pulses whose rate was 20,000 pulses per second. Measured width

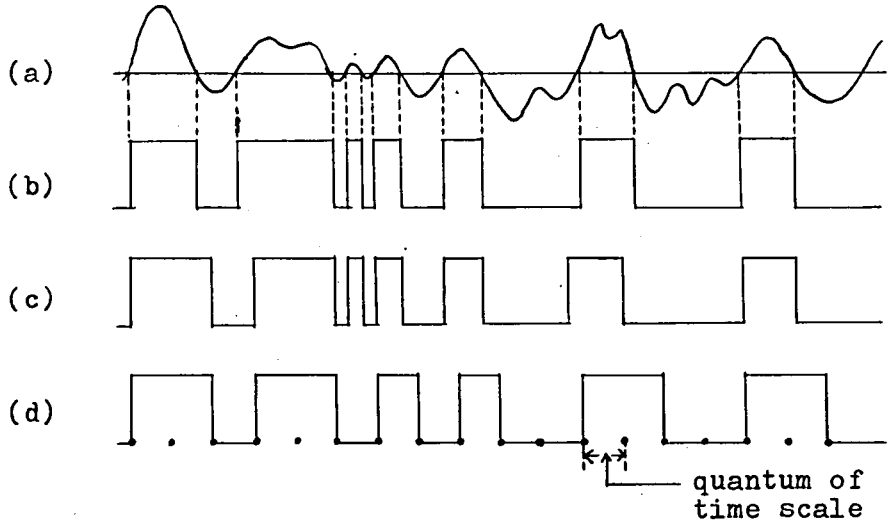


Fig. 5-1 Time quantized speech waves; (a) original speech wave, (b) infinitely clipped wave, (c) time quantized wave in the present experiment, (d) time quantized wave by method B in Licklider's experiment.

τ was transformed into one of intervals with predetermined widths in the following way. Now let Δf (cps) be a quantum in terms of frequency scale and integers a_n and b_n be decided with the following equalities.

$$a_n = \left[\frac{20000}{2n\Delta f} \right]$$

$$b_n = \left[\frac{20000}{2(n+\frac{1}{2})\Delta f} \right]$$

[ ] is Gauss's notation.

(a_{n+1}, a_n)	b_n	corresponding range of frequency (cps)
50 ~	100	200 ~
25 ~ 50	33	400 ~ 200
16 ~ 25	20	625 ~ 400
12 ~ 16	14	834 ~ 625
10 ~ 12	11	1,000 ~ 834
8 ~ 10	9	1,250 ~ 1,000
8	8	1,430 ~ 1,250
7	7	1,670 ~ 1,430
6	6	2,000 ~ 1,670
5	5	2,500 ~ 2,000
4	4	3,330 ~ 2,500
3	3	5,000 ~ 3,330
2	2	10,000 ~ 5,000
1	1	~ 10,000

Table 5-1 An example of the method of quantizing.

If $a_{n+1} < \tau \leqq a_n$, then width τ' of the transformed interval is equal to b_n. In the range of high frequency where integer n is so large that integer a_n is not different from integer a_{n+1}, if $\tau = a_n$, then $\tau' = b_n (= a_n)$. In other words, the quantizing of the width of an interval is not performed. An example of correspondence of a_n to b_n for the frequency quantum Δf of 200 cps is shown in table 5-1. In the experiments three kinds of frequency quanta of 50, 100 and 200 cps were used.

Phoneme category	sounds
Vowel and semi-vowel	a, i, u, e, o ja, ju, jo, wa
Unvoiced consonant	pa, pi, pu, pe, po ta, tʃi, tsu, te, to ka, ki, ku, ke, ko sa, ʃi, su, se, so ha, hi, hu, he, ho
Voiced consonant	ma, mi, mu, me, mo na, ni, nu, ne, no ra, ri, ru, re, ro ba, bi, bu, be, bo da, de, do ga, gi, gu, ge, go za, ʒi, zu, ze, zo

Table 5-2 List of monosyllabic sounds used for experiments

Intelligibility of the subjecting speech wave to the transformations just described, including infinite clipping only, was measured for sixty-seven monosyllabic sounds shown in Table 5-2. In the test, transformed speech sounds were arranged in random sequence made with the aid of shuffling of cards and presented to four male listeners who had not been trained in particular to hearing of zero crossing speech wave, although they devoted themselves to studies of speech sounds. Test sounds were pronounced by a male announcer.

5-2-2 Simulation procedure

(1) Simulation program. The whole block diagram for computer simulation of the experiments planned in the previous section is appearing in Fig. 5-2.

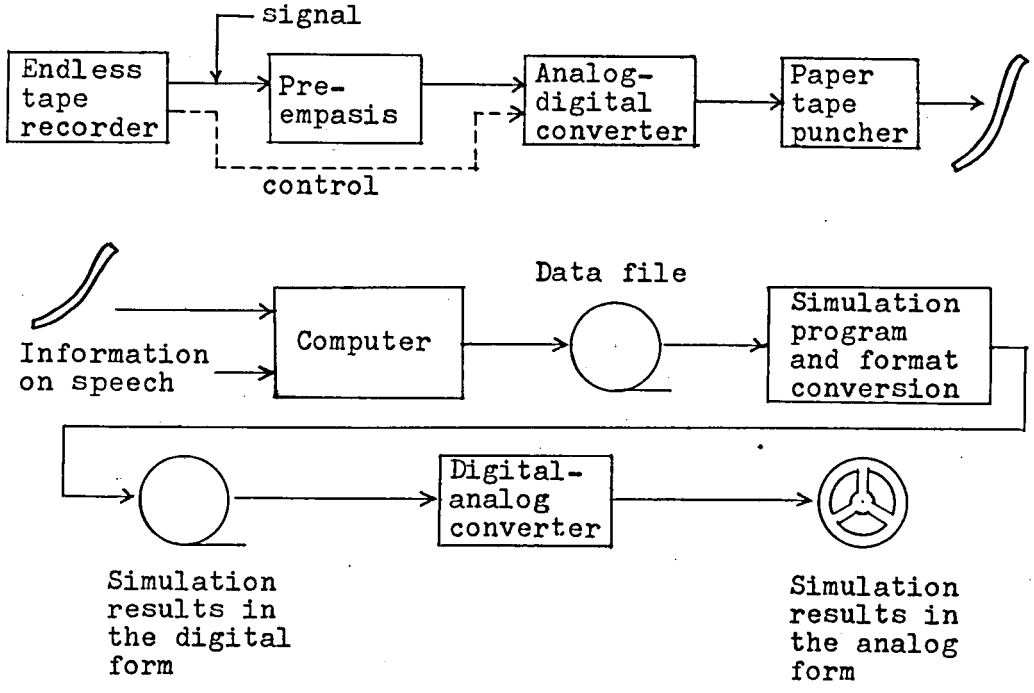


Fig. 5-2 Block diagram for simulation.

First simulation program will be explained while input and output procedures will be later. The block diagram for production of the time quantized zero corssing wave is shown in Fig. 5-3, which is, of course, expressed in terms of block types in KAIRO language. Its performance will be outlined below.

The original speech wave that is read in a computer through block INPMT (block type INP1) is transformed into infinitely clipped wave with Schmidt circuit OWV. Schmidt circuit has two slice levels as parameters, whose changes can produce different level crossing waves. Counter CNT is controlled by detecting positive going and negative going points with two triggering circuits TP1 and TP2 respectively. It measures the time length of an interval between two adjacent zero crossing points by counting a train of pulses generated by clock pulse generator CLOCK whose pulse rate is equal to the sampling rate of original sounds, that is, 20,000 pulses per second. In order to adjust the timing an input to the counter for resetting, the output of block OR, is passing through delay line ID1. Measured time length of the interval is introduced to an input terminal of quantizer QNT through block DA under the control of switch SWTCH whose output is the second input or the third input depending upon whether the first input is one or zero. Because a quantizer can accept only an analog type of signal, an output of the counter which is an integer type of signal is converted into an analog type of signal by block DA. Output of quantizer QNT gives the transformed time length of a zero crossing interval, which is used for reproduction of the time quantized zero crossing wave. Realization of the reproduction procedure with hardware is, of course, possible but is much complicated.

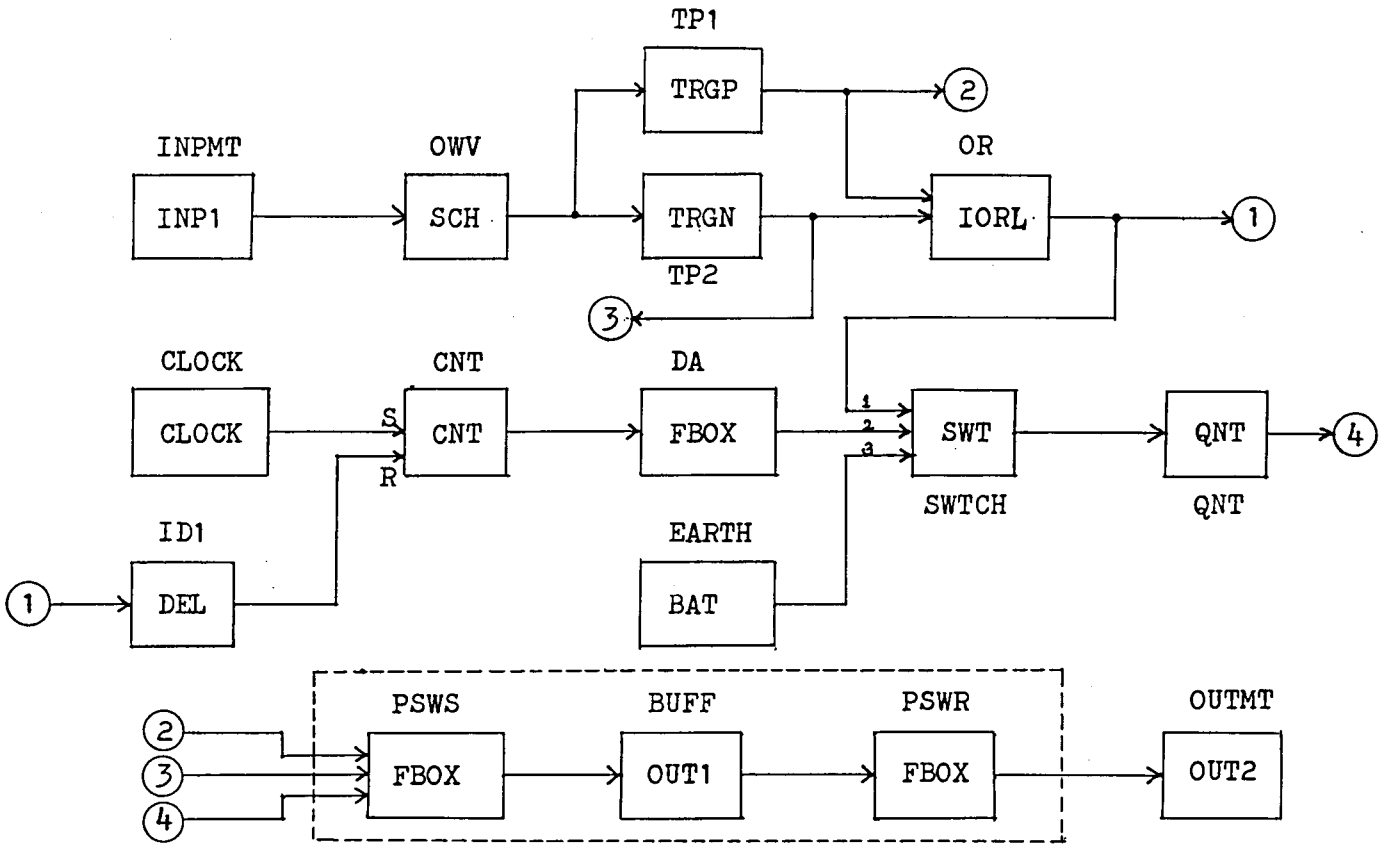


Fig. 5-3 Block diagram for generation of time quantized zero-crossing wave.

the constitution as shown in Fig. 5-3 is adopted, Furthermore, in order to spare a computing time for simulation and to demonstrate a tricky usage of block type FBOX provided for insertion of FORTRAN statements into KAIRO statements.

Blocks PSWS and PSWR make a pair so that the blocks bounded by them operate independently of the clock time controlling the whole system. In other words, from the moment that the output value of block QNT becomes not equal to zero, the other parts of the system stop operating but only the bounded parts begin to generate a rectangular wave whose width is equal to the output value of block QNT. While it is equal to zero, on the other hand, the bounded parts does not operate but the other parts do.

The time quantized zero crossing wave just generated feeds to block BUFF (block type OUT1) which is used for a temporary storage and records its input signals on a magnetic tape designated by one of its parameters. Block OUTMT (block type OUT2) does not operate before a simulation run is completed. It reads the signal recorded on the magnetic tape written by block type OUT1, normalizes all the values of the signal by the maximum of absolute values of the signal and records again the normalized signal on another magnetic tape as final simulation result.

In the block diagram shown in Fig. 5-3, different speech sounds can be transformed by changing the filename which is one of the parameters of block INPMT and different quantizations can be performed by changing the parameters of block QNT. KAIRO source program can be written in the form shown in Fig. 5-4, which can simulate the block diagram in Fig. 5-3 while changing input speech sounds.

(2) Input and output procedures. The speech sound to be studied is once transferred on an endless tape recorder which has two tracks.

```
#OXWQNT     KAIRO  C
C....TIME QUANTIZED ZERO-CROSSING WAVE....
 INPMT      INP1   30,NYOXQT0002,200,2
 OWV        SCH    0.1,-0.1"INPMT
 TP1        TRGP   "OWV
 TP2        TRGN   "OWV
 OR         IORL   "TP1,TP2
 ID1        DEL    "OR
 CLOCK      CLOCK
 CNT        CNT    1,200,200"CLOCK,,ID1
 DA         FBOX   1"CNT
      DA=FLOAT(CNT)
 EARTH      BAT    0.0"
 SWTCH      SWT    "OR,DA,EARTH
 PSWS       FBOX   8"QNT,TP1,TP2
      Q=QNT
      IF(TP2.EQ.1) GO TO 8000
      IF(TP1.EQ.1) GO TO 8001
      GO TO 8005
 8000 PSWS=1.0
      GO TO 8002
 8001 PSWS=-1.0
 8002 CONTINUE
 BUFF       OUT1   KANDA1,1,1,"PSWS
 PSWR       FBOX   3"BUFF
      Q=Q-1.0
      IF(Q.GE.0.0) GO TO 8002
 8005 CONTINUE
 OUTMT      OUT2   KANDA1,1,2,0"PSWR
 QNT        QNT    1.,1.,2.,2.,3.,3.,4.,4.,5.,5.,6.,6.,7./
                   7.,8.,8.,9.,10.,11.,12.,14.,16.,20.,25./
                   33.,50.,100."SWTCH
#           END
#           CHP
 INPMT             1,NYOXQT0003
 BUFF              1,AAA14"3,2
 OUTMT             1,AAA14"5,1
#           END
#OWARI      FIN
```

Fig. 5-4 The KAIRO program for generation of time quantized zoro-croossing wave.

The speech sound is recorded on one track and clock pulses of 20,000 pps, which are used for controlling the whole system consisting of an analog-digital converter and a paper tape puncher, on the corresponding portion of the other track. The speech wave is usually sampled at the rate of 20,000 cps and converted into a digital form in an accuracy of sign plus ten bits by the specially designed analog-digital converter. The paper puncher punches sampled values in the digital form on a paper tape at the speed of about 120 characters per second. The speech wave is reproduced repeatedly, converted and punched out little by little. Thus, the endless tape recorder acts as a buffer between the analog-digital converter and the paper tape puncher. The desired part of speech wave to be analyzed is determined by observing its wave form on a cathode ray tube and set manually on the analog-digital converter.

The speech data punched on a paper tape are read in a computer together with a filename, the number of records and the number of data in a record and recorded on a magnetic tape in the form which can be treated by block type in KAIRO language.

This speech data are processed by a simulation program and recorded again on another magnetic tape by the use of WRITE statement of FORTRAN. Since it cannot be accepted, as it is, by the digital-analog converter, its format must be converted so that it has no block gap in one speech wave. The conversion is conducted in the different phase from an execution of simulation program, because it must be done apart from the supervision of the operating system of the computer. The simulation result recorded continuously on the magnetic tape is reproduced and converted into an analog form by the digital-analog converter, which is

used for test sounds for an intelligibility test and other analysis, for instant, frequency analysis by the Sonagraph.

5-3 Intelligibility and confusion matrices

The intelligibility scores and the standard deviations of the monosyllabic sounds transformed as described above are shown in Table 5-3. Each sound was differentiated with R-C circuit before it was sampled and converted into the digital form. The time constant was 100 micro seconds which was determined by the result of preliminary experiments.

	intelligibility scores (%)	standard deviations (%)
a	62.3	5.6
b	67.5	3.8
c	70.9	4.9
d	70.9	3.6
e	95.5	2.4

Table 5-3 Intelligibility scores and standard deviations for various methods of quantizing, including infinite clipping only and for original sounds; (a) frequency quantum is 200 cps, (b) 100 cps, (c) 50 cps, (d) infinite clipping only and (e) original sounds.

In order to test the whole system, shown in Fig. 5-2, including the analog-digital converter, the paper tape puncher, the simulation program and the digital-analog converter, all the monosyllabic sounds used for tests were reproduced through the whole system without any transformation and their intelligibility was measured. It is also shown in Table 5-3, which is good enough to prove that the whole system operated correctly. The difference cannot be found between the infinite clipping only and the quantizing whose quantum is 50 cps in terms of frequency scale. In order to observe the effects of infinite clipping and time quantizing Sonagrams of the word /kanda/ are shown in Fig. 5-5.

The intelligibility scores are not so good as those of Licklider and Pollack.[31] This is partially because the words in PB lists were used in their experiment while the meaningless monosyllabic sounds were used in the present experiment and partially because our listener had not been trained in particular to hearing of zero-crossing speech sounds.

The results obtained with the experiment measuring the intelligibility were analyzed and the confusion matrices of various phonemes were constructed. They are shown in Table 5-4. Each position of the table contains the number of phonemes heard incorrectly. Since the confusion between voiced consonants and unvoiced consonants were hardly observed, they were grouped in seperate tables. Furthermore, semivowels were omitted from the tables because they were not confused at all. Monosyllabic sounds used for tests are in the form of CV (consonant + vowel). The confusion occurred chiefly in such a way that monosyllabic sounds CV was heard as monosyllabic sounds C'V. This was shown in the

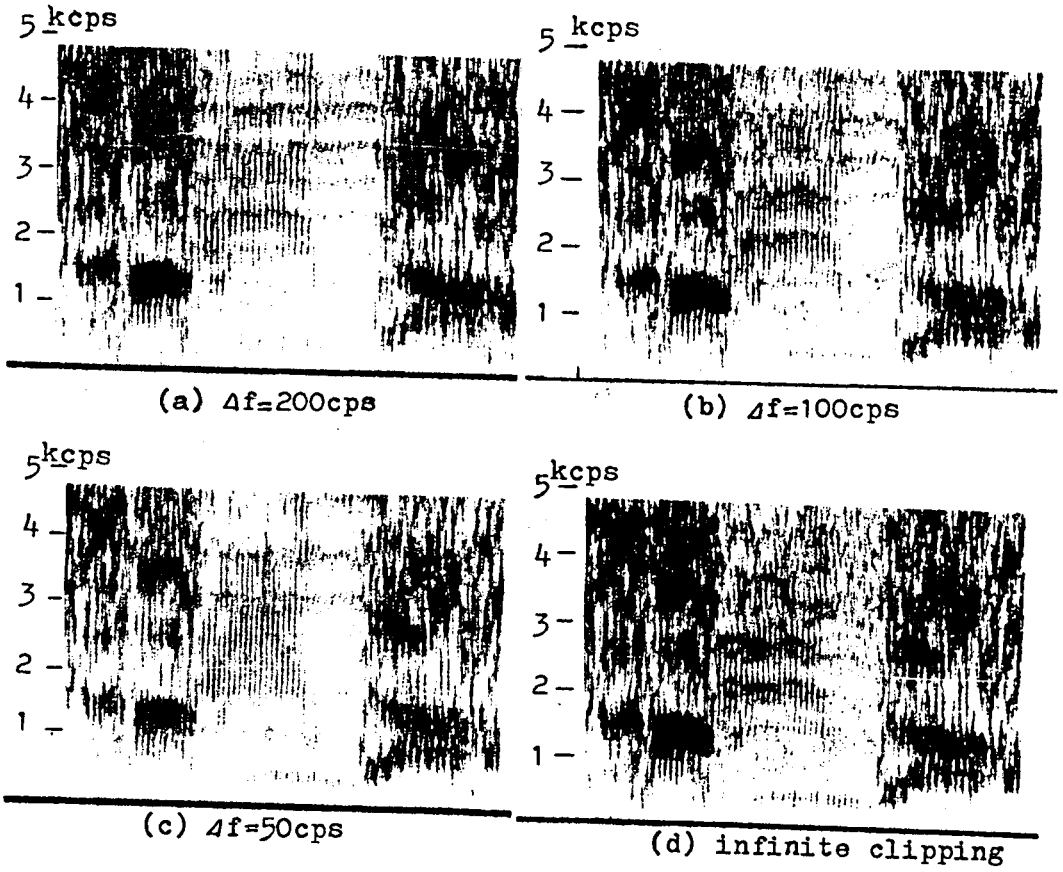


Fig. 5-5 Sonagrams of the word /kanda/ subjected to infinite clipping and time quantizing.

heard / spoken	not heard	p	t	k	h	s, ʃ	ts, tʃ	total
not spoken		2	2	1	2			7 (20)
p	6		2		1			9 (20)
t	2	3			1			6 (12)
k	1	1	3		1			6 (20)
h	2			13				15 (20)
s , ʃ							5	5 (20)
ts , tʃ				2				2 (8)

(a-1) unvoiced sounds

heard / spoken	not heard	m	n	r	b	d	g	z, ʒ	total
not spoken							2		2 (20)
m	1		7				1		9 (20)
n		1		3			1		5 (20)
r						5	1	1	7 (20)
b		6	3	1		2	1		13 (20)
d				4	2			1	7 (12)
g				2		2		2	6 (20)
z , ʒ							1		1 (20)

(a-2) voiced sounds

Table 5-4(a) Confusion matrices of phonemes for speech subjected to infinite clipping and time quantizing; frequancy quantum is 200 cps.

heard / spoken	not heard	p	t	k	h	s, ʃ	ts, tʃ	total
not spoken		1	1					2 (20)
p	8		1	1				10 (20)
t		3			1			4 (12)
k	1		1		3			5 (20)
h		2		13				15 (20)
s , ʃ					1		3	4 (20)
ts , tʃ				3				3 (8)

(b-1) unvoiced sounds

heard / spoken	not heard	m	n	r	b	d	g	z, ʒ	total
not spoken									0 (20)
m			6				1		7 (20)
n		2		1			2		5 (20)
r					1	4	1	1	7 (20)
b		4	4	1		2		1	12 (20)
d				1	2		1	2	6 (12)
g				1	1	3		1	6 (20)
z , ʒ							1		1 (20)

(b-2) voiced sounds

Table 5-4(b) Confusion matrices of phonemes for speech subjected to infinite clipping and time quantizing (continued); frequency quantum is 100 cps.

heard / spoken	not heard	p	t	k	h	s, ʃ	ts, tʃ	total
not spoken		1					1	2 (20)
p	5		2					7 (20)
t	1	3						4 (12)
k	1		1		3			5 (20)
h	2	3		9				14 (20)
s , ʃ							2	2 (20)
ts , tʃ				1				1 (8)

(c-1) unvoiced sounds

heard / spoken	not heard	m	n	r	b	d	g	z, ʒ	total
not spoken									0 (20)
m			4				2		6 (20)
n		2					2	1	5 (20)
r					4	1	2	1	8 (20)
b		6	3	1		1	1		12 (20)
d			3	2	1		1		7 (12)
g				1		2		1	4 (20)
z , ʒ									0 (20)

(c-2) voiced sounds

Table 5-4(c) Confusion matrices of phonemes for speech subjected to infinite clipping and time quantizing (continued); frequency quantum is 50 cps.

heard / spoken	not heard	p	t	k	h	s, ʃ	ts, tʃ	total
not spoken		2		1				3 (20)
p	6		1					7 (20)
t		3			1			4 (12)
k	2				3			5 (20)
h	1			11				12 (20)
s , ʃ							2	2 (20)
ts , tʃ				1				1 (8)

(d-1) unvoiced sounds

heard / spoken	not heard	m	n	r	b	d	g	z, ʒ	total
not spoken									0 (20)
m			9				1		10 (20)
n		1					3	1	5 (20)
r			1		2	2	1		6 (20)
b		4	5	1		1	2	1	14 (20)
d			1	1	2			2	6 (12)
g				1				1	2 (20)
z , ʒ									0 (20)

(d-2) voiced sounds

Table 5-4(d) Confusion matrices of phonemes for speech subjected to infinite clipping and time quantizing (continued); infinite clipping only.

tables as the confusion between phoneme /c/ (spoken) and phoneme /c'/ (heard). If CV were heard as CV', the confusion would be shown in the main diagonal of each table. The confusion contained in row "not spoken" means that vowels were heard as the sounds in the form of consonant + vowel, and the confusion contained in column "not heard" means that CV was heard as vowels only. The last column of each table contains the total heard incorrectly for each phoneme and the number of phonemes used for tests in brackets. The contents of positions having no entry are zero.

The effect of infinite clipping on the confusion of phonemes is different from the results obtained with the experiments conducted with physical devices, in which amplified and clipped noise filled the silent interval. It was reported[36] that this noise merged with the initial burst of the stop consonant and increased the confusion between voiced stop consonants and semivowels, and between unvoiced stop consonants and fricative consonants. Since noise in the silent interval was suppressed in the present experiment when the speech wave was sampled and converted into the digital form by the analog-digital converter, the interference of clipped noise cannot occur. No existence of clipped noise in silent interval causes the initial parts of all the sounds to sound like plosives. The confusion that sound /h/ was heard as sound /k/ and unvoiced fricative as unvoiced africative is due to this fact. The initial burst of unvoiced stop consonant is characterized with its sharp explosive and its isolation followed by an aspiration of low amplitude. The mutual confusion between unvoiced stops and vowels is primarily due to the fact that infinite clipping uniforms the amplitude of the initial burst and the aspiration. It seems to be caused by

this fact that the longer aspiration has a sound, the less it is confused within phoneme category of unvoiced stops or that of voiced stops.

As expected, the larger is the quantum in terms of frequency scale, the greater effect of quantizing appears on the intelligibility scores and confusion of phonemes. As the present methods of quantizing give the change to the frequency spectrum in the range of less than 1500 cps as shown in Table 5-1, the great influence appears in vowellike sounds which have formants as an important cue in the corresponding range of frequency. It is inferred that vowels, when heard incorrectly, are more often heard as unvoiced stops than as any other phonemes when the quantizing changes the formant frequency of a vowel towards the formant locus of an unvoiced stop.

5-4 Conclusion

In this chapter, one of applications of KAIRO language to speech information processing was reported. In these experiments, the speech wave was subjected to infinite clipping of the amplitude and quantizing of the time length of an interval between two adjacent zero crossing points. The methods of quantizing were linear in terms of frequency scale in which the frequency quanta of 50, 100 and 200 cps were used. Intelligibility of the subjecting speech wave to such transformations was measured and confusion matrices of phoneme were constructed for various methods of quantizing.

The experiments were simulated by means of a computer. The simulation program was written in KAIRO language which had been developed in chapters three and four. Since programming was easier with KAIRO

language than with other general purpose language, the merit of KAIRO language seems to have been proved sufficiently.

CHAPTER 6

A procedure for automatic extraction of formant region —— an application of KAIRO language to speech information processing (II)[37][38]

6-1 Introduction

When speech sounds are analyzed by means of a frequency analyzer, some peaks appear in their spectra. Especially in vowel sounds, four dominant peaks are appearing in the frequency range lower than about 5 kcps. As mentioned in chapter five, the frequencies at these peaks of the spectrum are called formant frequencies and they are named first formant, second formant, etc., in the order they occur in the frequency scale and nearly corresponding to the resonance frequencies of the vocal tract cavity system; that is, the acoustic tube from lips to the glottis. The lowest two or three formants contribute primarily to the discrimination among vowels. At an attempt to recognize the speech sounds automatically, therefore, the first two or three formants play the important roles in vowel sounds. Although a number of methods of extracting formants[39][40][41][42] have been studied by means of digital computers as well as various analog techniques, the method is not discovered by which the accurate formants are extracted automatically and in real time.

As an approach to the goal, in this chapter, a procedure was studied for extracting the frequency regions, where the formants exist, rather than the accurate formant frequencies.

This procedure was studied independently of the development of KAIRO language but was nevertheless done by using the computer program

and some successful results were obtained. Realization of the procedure with hardware devices through analog technique was tried by the use of the block types in KAIRO language in order to obtain an estimate of the scale of whole system. The supplementary experiments which were performed using the block diagram constructed increased the reliability of the results previously obtained.

6-2 Procedure and simulation program

6-2-1 Procedure

As mentioned in the preceding section, the formant frequency is frequency at which the dominant peak appears in a vowel spectrum. If the formant amplitudes, or the amplitudes of the dominant peaks in a spectrum, are nearly equal, the formant regions are expected to be extracted as the frequency regions where spectral amplitudes are larger than a certain threshold value. In general the formant amplitudes, however, are not equal in a vowel spectrum. Especially the second formant amplitude of a Japanese vowel /u/ is often lower than other formant amplitudes. The change of the input level of a vowel sound and of the gain of the frequency analyzing system involves the change of a spectral level. In extracting the formant regions by the threshold method, therefore, it is not suitable to use the constant threshold all over the frequency range of interest. The threshold satisfying the following requirements is desirable;

(1) it is large or small depending on the local amplitude of a spectrum,

(2) it is relatively invariant to the variation of a spectral level,

in other words, the formant regions to be extracted are not changed by it.

A locally weighted mean was adopted as the threshold, since it satisfies these requirements as shown below. The threshold curve was computed for a spectrum expressed on a linear scale.

Let S(i) be the value of the i-th channel of amplitude spectrum and T(i) be the corresponding threshold. They are both represented on a linear scale. The formant regions are defined as the frequency regions in which the difference between the spectrum and its threshold is positive. Difference D(i) of the i-th channel is computed on a decibel scale and threshold T(i) is computed in the form of a locally weighted mean of power spectrum $S(i)^2$. Thus,

$$D(i) = 20 \log S(i) - 10 \log T(i) \qquad (i = 1, 2, \ldots, n) \qquad (1)$$

$$T(i) = \sum_{j=c_1}^{c_2} w_{ij} S(j)^2 \qquad (i = 1, 2, \ldots, n) \qquad (2)$$

where c_1 is the larger of two integers zero and i-r, c_2 is the smaller of two integers n and i+r, n is the number of channels of the frequency analyzer used for obtaining the amplitude spectrum, fifty in the present case and r is a given positive integer. The w_{ij} are weighting coefficients of the j-th channel for the calculation of the threshold of the i-th channel and are expressed as follows;

$$w_{ij} = \begin{cases} a_i / \left\{ 1 + \dfrac{(i-j)^2}{k^2} \right\} & (i \neq j) \\ 0 & (i = j) \end{cases} \qquad (3)$$

where a_i and k are parameters. The w_{ii} were equated to zero so that the difference between S(i) and T(i) might be emphasized in the formant

regions. When the parameter k is small, a threshold curve closes to its original spectrum. Since the threshold curve is quite coincident with its original spectrum at the limit of k tending to zero, it is no more useful as a threshold curve. On the other hand, the threshold curve becomes gradually flat, as k increases. Since the weighting coefficients are equal at the limit of k tending to infinite, the threshold curve does not satisfy the first requirement mentioned above. Although it is sure from these considerations that the optimum value of the parameter k exists, it is difficult to find it thoretically. So the parameter k was set three from the result of the test computation of the threshold curves for some spectra in the stationary and the transient parts of connected vowel sounds. The a_i are selected such that $\sum_{j=C_1}^{C_2} w_{ij} = 1$.

In the present case, the variation of the spectral level means the multiplication of S(j) by a constant factor A. It is easily shown by substituting AS(j) into equations (1) and (2) that difference D(i) is not changed. The difference between a spectrum and its threshold is turned to be invariant to the variation of the spectral level; in other words, the formant regions to be found can be invariant since they are extracted depending only on the sign of the difference. Of course, T(i), locally weighted mean of S(j), is large or small depending on the local amplitude of the spectrum. Thus it is known that T(i) defined by equation (2) satisfies the two requirements mentioned above.

6-2-2 Block diagram for simulation

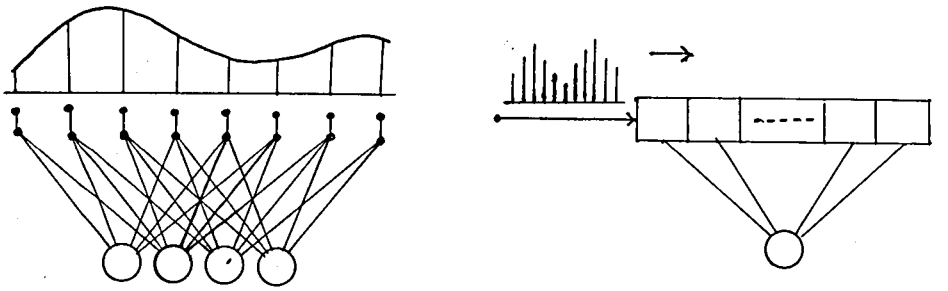
In general, the frequency spectrum of speech wave can be obtained

with a bank of filters. Outputs of the filters are rectified, smoothed by low pass filters and form a running spectrum.

There are two methods of computing the threshold using this spectrum. The first is the parallel computing with the circuit consisting of a number of resistors connected with output terminals of the bank of filters as shown in Fig. 6-1(a). The value of each resistor corresponds to the value of the weighting coefficient appearing in equation (3). The second is the serial computing with the circuit composed of a multi-tap delay line or a shift register and a few resistors connected with the taps as shown in Fig. 6-1(b). In this method, the outputs of the bank of filters are once held, sampled and scanned by a multiplexor. The spectrum which is converted into a serial form is led to the weighting circuit and the threshold is computed serially.

The latter was adopted here for the purpose of decreasing the number of devices. Since a computing time taken to simulate the bank of filters increases as the number of filters becomes larger, the frequency spectrum of speech wave is obtained by means of actual filters which will be described in detail later. Instead of simulating the bank of filters, the spectrum which is punched on a paper tape by the instruments described in chapter five is read in a computer. The timing of reading the spectum is controlled by the signal which is to control the multiplexor acctually.

On these conditions, the system extracting formant regions were simulated by the use of KAIRO language. The block diagram of the whole system is appearing in Fig. 6-2. It begins to operate by setting on manual switch ISRT (block type FBOX). Under the control of the signal generated by blocks included in the part denoted in Fig. 6-2



(a) Parallel computation (b) Serial computation

Fig. 6-1 Methods of computing thresholds.

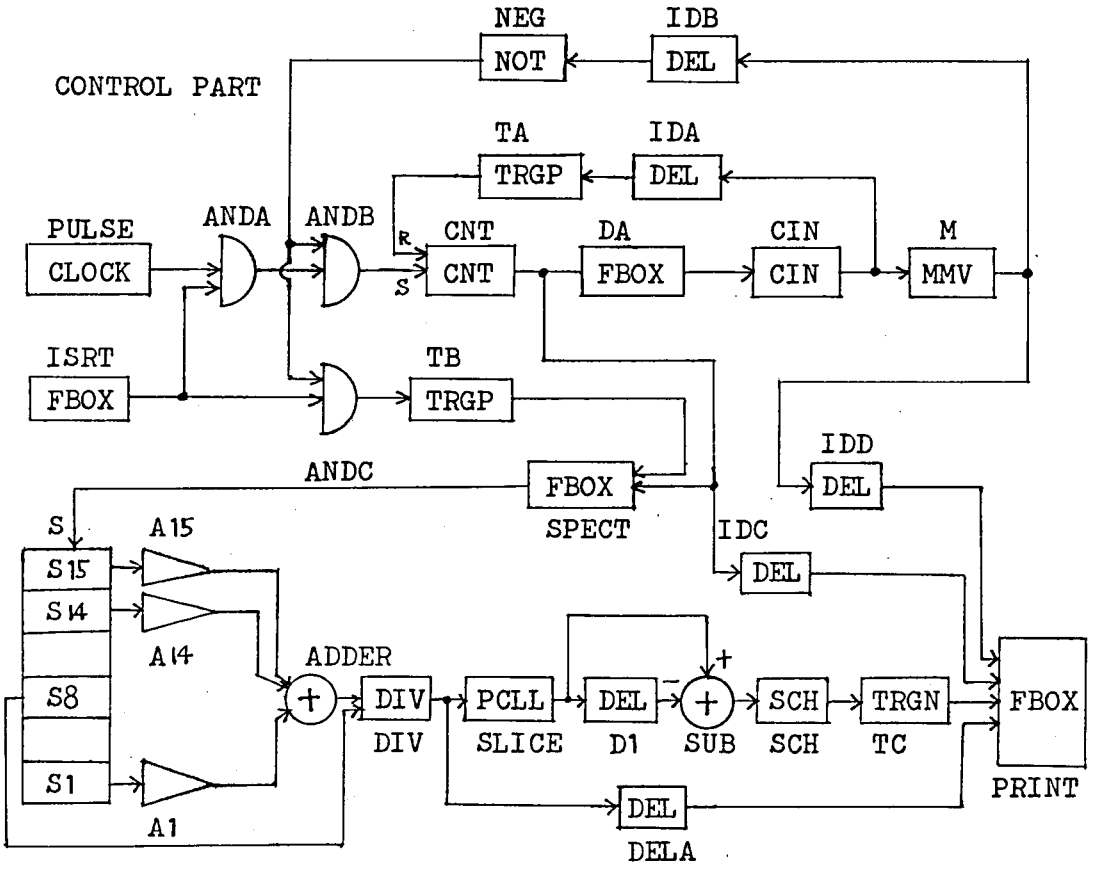


Fig. 6-2 Block diagram for extraction of formant regions.

as CONTROL PART, block SPECT (block type FBOX) which reads the spectrum in a computer and converts it into a serial form. The serial spectrum is led to the weighting circuit consisting of shift register S whose cells are named S1, S2, ..., S14 and S15, and amplifiers A1, A2, ..., A14 and A15. The number of cells is related to parameter r, the width of weighting range, appearing in the previous section; that is, 2r+1 is equal to the number of cells. The threshold of the first channel of the bank of filters appears as an output of adder ADDER in seven clock times after the serial spectrum is led to the shift register. Thus the threshold is in turn computed every clock time.

The next serial spectrum must begin to be led to the shift register in eight clock times after the last channel of the current spectrum is led to it in order that the computing of thresholds of the current spectrum cannot be disturbed by the next spectrum. The signal controlling this timing is also generated in the control part. Its performance is as follows. Counter CNT counts a train of pulses and its output corresponds to the channel number of the spectrum to be read in. When it is coincident with the number of channels, coincidence circuit CIN produces the signal to trigger monostable multivibrator M, which controls the timing mentioned above. Its pulse width is equal to the period of seven clock times and during this period the input gate of the counter is inhibited. Outputs of the counter and monostable multivibrator are used for displaying the results too. The time chart of this part is shown in Fig. 6-3.

The difference between the spectrum and its threshold is printed out in the same form as the pattern of Sonagram by block PRINT (block

type FBOX). As it must be displayed on a decibel scale, the ratio of the spectrum to its threshold is computed by block DIV, and the conversion of it into a decibel scale is performed in block PRINT. In order to emphasize the maximums of the differences in the formant regions they are differentiated spacially by blocks SUB and D1 and the channels giving the maximums are detected by block SCH.

KAIRO source program which is converted from Fig. 6-2 is shown in Fig. 6-4. Subroutine INPUT which is called here reads the spectrum in a computer and PRINT prints out the pattern of the difference.

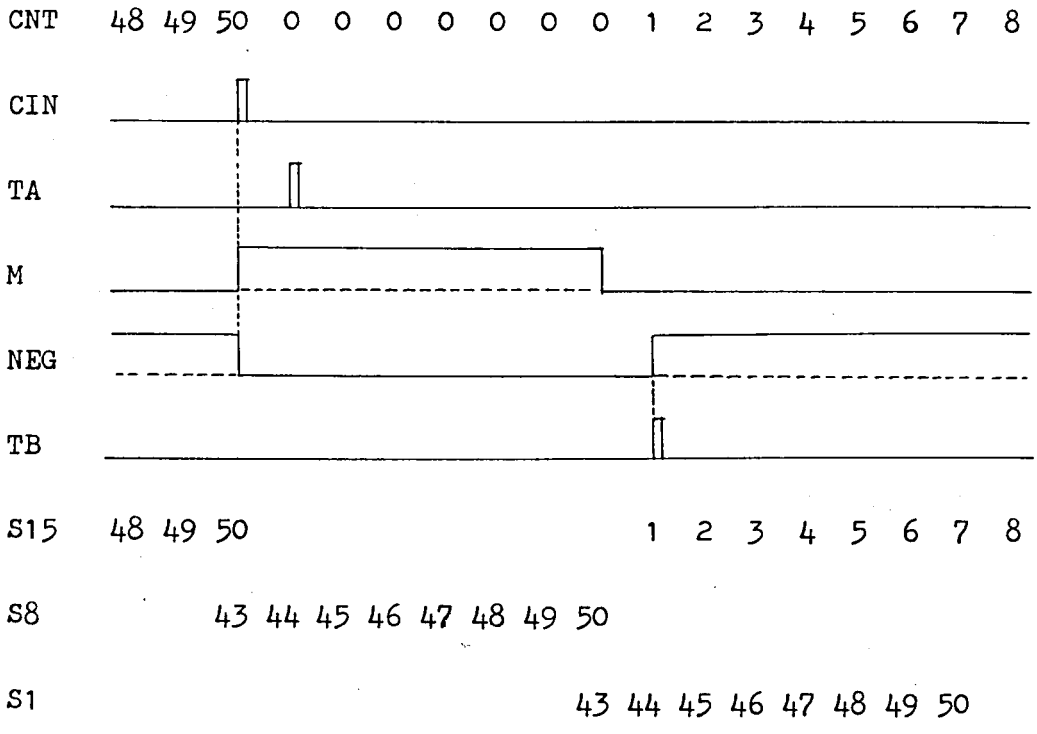


Fig. 6-3 Time chart of the block diagram shown in Fig. 6-2.

```
#A0833     KAIRO  C
C....EXTRACTION OF FORMANT REGIONS....
C....CONTROL PART....
 PULSE      CLOCK
 ISRT       FBOX   1
      ISRT=1
 ANDA       ANDL   "PULSE,ISRT
 ANDB       ANDL   "ANDA,NEG
 ANDC       ANDL   "ISRT,NEG
 CNT        CNT    0,100,100"ANDB,,TA
 TA         TRGP   "IDA
 IDA        DEL    "CIN
 DA         FBOX   1"CNT
      DA=FLOAT(CNT)
 CIN        CIN    49.5,50.5"DA
 M          MMV    7"CIN
 IDB        DEL    "M
 NEG        NOTL   "IDB
 TB         TRGP   "ANDC
C....COMPUTATION OF THRESHOLDS....
 SPECT      FBOX   1"CNT,TB
      CALL INPUT(SPECT,CNT,TB)
 S          SHR    "SPECT
 S1         DUM    "S
 S2         DUM    "S
             :
             :
 S8         DUM    "S
             :
             :
 S15        DUM    "S
 A1         AMP    0.02525"S1
 A2         AMP    0.03255"S2
             :
             :
 A15        AMP    0.02525"S15
 ADDER      ADD    "A1,A2,....,A7,A9,....,A15
C....DISPLAY....
 DIV        DIV    "A8,ADDER
 SLICE      PCLL   1.0"DIV
 D1         DEL    "SLICE
 SUB        SUB    "SLICE,D1
 SCH        SCH    0.1,-0.1"SUB
 TC         TRGN   "SCH
 DELA       DEL    "DIV
 IDC        DEL    8"CNT
 IDD        DEL    8"M
 PRINT      FBOX   1"DELA,TC,IDC,IDD
      CALL PRINT(DELA,TC,IDC,IDD)
#           END
#           FIN
```

Fig. 6-4 The KAIRO program for extraction of formant regions.

6-2-3 Spectral data

The speech wave to be analyzed is once recorded on an endless tape recorder. It is reproduced repeatedly and analyzed by a frequency analyzer. The short time spectrum obtained is sampled, scanned by a multiplexor and converted into the digital form by means of the instrument described in chapter 5.

The frequency analyzer[43] is composed of thirty channels. Each channel consists of a single tuned filter, an amplifier, a rectifier and a low pass filter. The single tuned filter has a half power bandwidth of 33 cps and the spacing of their center frequencies is uniformly 33 cps. This bank of filters can cover the frequency range of 13 kcps through 14 kcps. The frequency range of speech wave is shifted to this range by modulation. The analyzing range of speech wave is determined by the selection of the carrier frequency of modulation. The analysis is conducted on each 1 kcps for each revolution of the endless tape recorder in such a way as 0-1 kcps, 1-2 kcps, ... , 7-8 kcps for corresponding carrier frequency 14 kcps, 15 kcps, ... , 21 kcps respectively.

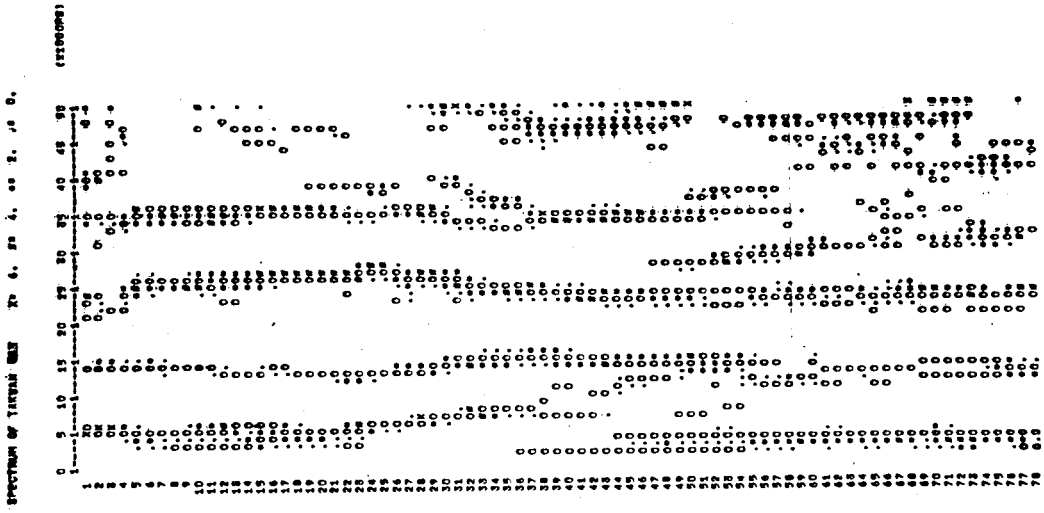
Since vowellike sounds were chiefly used in this experiment, the analysis was limited at the range of 0-5 kcps. Thus the spectrum of 150 channels was obtained. However, the number of channels was so large that it was incorporated into the spectrum of 50 channels by summing up outputs of three successive channels in a power dimension in a computer. This power spectrum was used for the experiment of extracting formant regions.

6-3 Results and conclusion

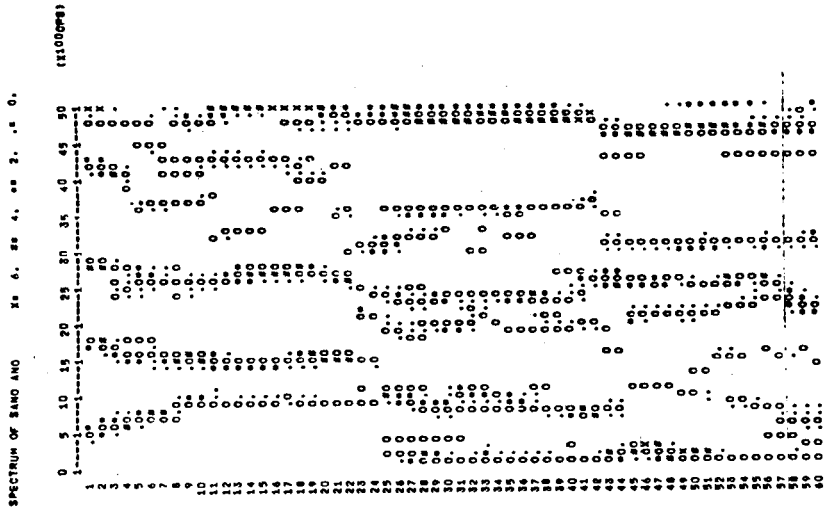
Difference spectra D(i) were computed for the complete spectra obtainsed from vowellike sounds. The frequency regions in which they had positive values were extracted as the formant regions. The difference spectrum was quantized with a 2 dB step in the formant regions. Corresponding to the complete spectrum of the original sound, the difference spectrum was printed out. An appropriate letter was assigned for each quantized level in the formant regions and no letter were printed out for the regions in which the differences were negative.

The typical results of this procedure are shown in Fig. 6-5(a) and (b). Symbol "X" denotes the quantized level higher than 6 dB, "#" from 6 dB to 4 dB, "*" from 4 dB to 2 dB, and "." from 2 dB to 0 dB. Symbol "O" denotes the channels of peaks in extracted formant regions. Spectral data in these figures were obtained from sounds /uan/ of word /takuan/ and /ano/ of word /sano/ which were spoken by a male talker. The first formant region, the second formant region and the third formant region in Fig. 6-5(a) are extracted clearly, though low harmonic components are also extracted in some parts. The movements of formant regions are apparent as much as in the pattern of the Sonagram. The formant regions may be said to be extracted fairly well for tested sounds, although they are rather few in number.

The formant regions obtained by this procedure may be used effectively as the first approximations for the accurate formant extraction method, say, Analysis-by-Synthesis method[39][40] which has been proposed and developed by Stevens, et al. Because of not requiring so much time, this method may be useful and substituted for the Sonagram



(a) /uan/ of the word /takuan/.



(b) /ano/ of the word /sano/.

Fig. 6-5 Typical results.

if it is desired to deal with a great number of data and is not required high accuracy in extracting formants. Since the procedure could be realized with simple analog circuits as shown in Fig. 6-2, it might be expected to play an important role as a part of a real time processing system of speech sounds.

CHAPTER 7

Conclusion

In this thesis, a simulation language, KAIRO, was described and its applications to speech analysis were reported. KAIRO language is suited for the simulation of information processing systems, including both analog and digital signals, such as character recognition systems, speech analysis and synthesis systems and communication networks.

The detailed structure of KAIRO language was explained in chapter 3. A simulation model can be described in the form of a block diagram familiar to engineers. It can be constructed by block types provided as basic functions which are characteristic in the problems mentioned above. Furthermore the programmer can define his desirable functions with FORTRAN language. A statement appearing in KAIRO source program corresponds to one block in the block diagram and the sequence of statements may be independent of the flow of signals. The extreme of its advantages is this ability of automatic sequence control which is committed to the care of the programmer in a general purpose language.

In chapter 4, the KAIRO compiler was described in detail. It is divided into two main parts. One analyzes the syntax of a source program and the other interprets block types included in it and converts them into the corresponding sets of FORTRAN statements. The structure of KAIRO language is essentially the representation of a block diagram in terms of the list structure. Therefore the push down store is effectively used for the syntactic analysis of the KAIRO program. The interpretation of block types is performed in the general form. The

F-Table which contains information necessary for analysis of a statement and the statement mould which is the original pattern for interpretation of block type are provided for each block type. Object statements can be obtained simply by replacing the dummy variables contained in the mould by the actual ones specified by the KAIRO statement. The addition of a new function to the KAIRO language as block type can be performed so easily that only the F-Table and the statement mould may be added to the compiler.

Effectiveness of KAIRO language has been proved through its applications to the simulation problems of speech analysis in chapters 5 and 6. KAIRO language seems to provide the method of converting a simulation model into a computer program. Judging from author's experience in programming with FORTRAN language, a computing speed of the object program generated by the KAIRO compiler seems to be as fast as that of the program coded directly in FORTRAN language. On the other hand, the object program tends to be large in comparison with the program directly coded. For instance, the circuits composed of a shift register and amplifiers appearing in chapter 6 should be converted into a short and simple program including a DO statement, but in fact the same pattern of program is repeated the same number of times as that of the amplifiers. This is because the policy of the KAIRO compiler consists in that a computing time is made as short as possible.

The design of KAIRO language was planned in April of 1965. The computer, HITAC-5020, which was used for its development was opened to the public users in the Kyoto University Computation Center in Octover of 1965. Its core memory of 16 K words was insufficient for a scale of the compiler. It was, therefore, obliged that the core resident

part of the compiler was so small that it could accept a block diagram including as many blocks as possible. This fact produced various kinds of disadvantages on the construction of the compiler. The extreme of them was that the handling of a magnetic tape was impossible because the system subroutine for it required a large scale of memory. The core memory of 32 K words could accommodate the whole compiler program simultaneously and would enable the object program to be written on a magnetic tape.

The KAIRO compiler was completed in March of 1967 after full two years. This period does not seem so long as compared with a scale of the compiler. However, there is no doubt that the facts that the computation center has been managed completely under a closed shop and that a turn around time becomes longer because of a tremendous demands for a large scale and high speed computer have made it greatly difficult and inefficient to develop the programming system with an assembly language.

ACKNOWLEDGMENTS

The author wishes to express his hearty gratitude to Professor Toshiyuki Sakai for his constant support, understanding and encouragement during the course of this study.

He also wishes to thank Assistant Professor Shuji Doshita for his constant guidance and earnest discussion on the design of the simulation language and the study of speech analysis.

He is indebted to Mr. K. Tabata, Mr. K. Otani and Mr. M. Matsushita for their co-operations and helps in the experiments of chapter 5.

The author wishes to thank staffs of Professor T. Sakai's laboratory, especially Dr. M. Nagao and Dr. H. Nishio for their daily discussions and useful suggestions during his study. Staffes of the Kyoto University Computation Center helped him in utilizing the computer KDC-II (HITAC 5020) and solving some troubles in the development of the KAIRO compiler.

BIBLIOGRAPHY

(1) W. P. Heising, "History and Summary of FORTRAN Standardization Development for the ASA," Comm. ACM, Vol. 7, 590-625 (1964).

(2) P. Naur, et al., "Report on the algorithmic language ALGOL 60," Comm. ACM, Vol. 3, 299-314 (1960).

(3) "IBM System/360 Operating System PL/1 Language Specification," IBM Syst. Reference Library Form No. C28-6571-4 (1966).

(4) Yu. A. Shreider, et al., "THE MONTE CARLO METHOD," (Pergamon Press, London, 1966) Translated from Ю.А. Шрейер, "Метод статистических испытаний — Метод монте-карло," (Физматгиз, Москва 1962)

(5) Richard E. Dauson, "Simulation in the Social Science," Simulation in Social Science: Readings, by Harold Guetzhow (ed.), (Prentice-Hall, Inc., 1962).

(6) Harold Guetzhow,"A Use of Simulation in the Study of Inter-nation Relations," Behavioral Science, Vol. 4, 183-191 (1959).

(7) G. Gordon, "A General Purpose Systems Simulator," IBM Systems Journal I (Sept., 1964).

(8) A. L. Pugh, "DYNAMO User's Manual," (The M.I.T. Press, 1963).

(9) H. M. Markowitz, et al., "SIMSCRIPT, A Simulation Programming Language," (Prentice-Hall, 1963).

(10) J. L. Kelly, et al., "A Block Diagram Compiler," Bell Syst. Tech. J., Vol. 40, 669-676 (1960).

(11) J. W. J. Williams, "E.S.P. The Elliott Simulation Package," The Computer Journal, Vol. 6, 328-331 (1964).

(12) J. G. Laski, et al., "Control and Simulation Language," The Computer Journal, Vol. 5, 194-198 (1962).

(13) R. W. Conway, et al., "CLP — The Cornell List Processor," Comm. ACM, Vol. 8, 215-216 (1965).

(14) D. H. Kelley, et at., "Montecode — an interpretive program for Monte Carlo simulations," The Computer Journal, Vol. 5, 88-93, (1962).

(15) C. C. Holt,"Program SIMULATE, a User's and Programmer's Manual," (Univ. Wisconsin, 1964).

(16) K. D. Tocher, "Review of Simulation Languages," Operational Research Quarterly, Vol. 16, 189-217 (1965).

(17) T. Sekine, et al., "On Simulation Language," J. Information Processing Society of Japan, Vol.6, 30-38 & 89-95 (1965).

(18) R. M. Golden, "Digital Computer Simulation of Sampled Data Communication Systems Using the Block Diagram Compiler: BLODIB," Bell Syst. Tech. J., Vol. 45, 345-358 (1966).

(19) T. Sakai, S. Doshita and Y. Niimi, "Block Diagram Simulation System," Record of the 1966 Joint Convention of the KANSAI District of IECEJ*.

(20) T. Sakai and Y. Niimi, "Programming System for Block Diagram Simulation," Record of the 1966 Joint Convention of the IECEJ*.

(21) T. Sakai and Y. Niimi, "Programming System for Block Diagram Simulation," Technical Report of the Professional Group on Electronic Computer of IECEJ*, (Jan., 1967).

(22) J. L. Dineley, et al., "KALDAS, an algorithmically based digital simulation of analogue computation," The Computer Journal, Vol.9, 181-187 (1966).

(23) M. M. Baum, et al., "Use of digital analogue simulator (DAS)," The Computer Journal, Vol. 9, 175-180 (1966).

(24) S. Matoba, "Simulator with an Analog Oriented Input Language (MDAS)," Jour. IECEJ*, Vol. 49, 1351-1356 (1966).

(25) K. Takashima, et al., "A Universal Program System Simulating Logic," Jour. Information Processing Society of Japan, Vol. 4, 64-72 (1963).

(26) "Program Manual HITAC 5020 Assembly System (HISAP)," (Hitachi, Ltd., Tokyo, 1965).

(27) "Program Manual HITAC 5020 FORTRAN (HARP)," (Hitachi Ltd., Tokyo, 1965).

(28) C. J. Shaw, "Assemble or Compile ?" Datamation Vol. 12, 56-62 (Sept., 1966).

(29) G. Fant, "Acoustic Theory of Speech Production," (Mouton & Co., Hague, 1960).

(30) J. L. Flanagan, "Speech Analysis Synthesis and Perception," (Springer-Verlag, Berlin, 1965), p.244.

(31) J. C. R. Licklider and I. Pollack, "Effects of Differentiation, Integration, and Infinite Peak Clipping upon the Intelligibility of Speech," J. Acoust. Soc. Am., Vol. 20, 42-51 (1948).

(32) P. Marcou and J. Daguet, "New Methods of Speech Transmission," Information Theory Third London Symposium, 231-244 (1955).

(33) K. Hiramatsu, "Zero-crossing Information of S.S.B. Speech Signal," J. Acoust. Soc. Japan, Vol. 18, 301-309 (1962).

(34) K. Hiramatsu, "Speech Band Compression System Using S.S.B.-Clipping — Formac — ," J. Acoust. Soc. Japan, Vol. 18, 310-319 (1962).

(35) J. C. R. Licklider, "The Intelligibility of Amplitude-Dichotomized, Time-Quantized Speech Wave," J. Acoust. Soc. Am., Vol. 22, 820-823 (1950).

(36) W. A. Ainsworth, "Relative Intelligibility of Different Transforms of Clipped Speech," J. Acoust. Soc. Am., Vol. 41, 1272-1276 (1967).

(37) T. Sakai, S. Doshita and Y. Niimi, "Extraction of Formant Regions by a Locally Weighted Mean Curve," Record of the 1964 Joint Convention of the IECEJ*.

(38) T. Sakai, S. Doshita and Y. Niimi, "A Procedure for Formant Domain Extraction," Studia Phonologica Vol.IV, 90-95 (1960).

(39) K. N. Stevens, "Toward a Model for Speech Recognition," J. Acoust. Soc. Am., Vol. 32, 47 (1960).

(40) A. P. Paul, et al., "Automatic Reduction of Vowel Spectra: An Analysis-by-Synthesis Method and Its Evaluation," J. Acoust. Soc. Am., Vol. 36, 303 (1964).

(41) M. V. Mathews, et al., "Pitch Synchronous Analysis of Voiced Sounds," J. Acoust. Soc. Am., Vol. 33, 179-186 (1961).

(42) W. Lawrence, "Formant Tracking by Self Adjusting Inverse Filtering," Paper C5, Stockholm Speech Communication Seminar (1962).

(43) S. Doshita, "Studies on the Analysis and Recognition of Japanese Speech Sounds," (Doctoral Thesis, Kyoto University, 1965), chapt. 3 of PART I.

IECEJ*: The Institute of Electrical Communication Engineers of Japan.